Diplomarbeit

Realisierung von Fluchtwegsdaten im IFC-Format mithilfe von IfcOpenShell

ausgeführt zum Zwecke der Erlangung des akademischen Grads
Diplom-Ingenieur
eingereicht an der TU Wien, Fakultät für Bau- und Umweltingenieurwesen

---

Diploma Thesis

Realization of escape route data in the IFC format using IfcOpenShell

submitted in satisfaction of the requirements for the degree
Diplom-Ingenieur
of the TU Wien, Faculty of Civil and Environmental Engineering

Matthias Haselberger, BSc

Matr.Nr.: 01526763

Betreuung: Associate Prof. Dipl.-Ing. Dr.techn. **Christian Schranz**, M.Sc.
Univ.Ass. Dipl.-Ing. **Simon Fischer**, BSc
Univ.Ass. Dipl.-Ing. Dr.techn. **Harald Urban**, BSc
Institut für Baubetrieb und Bauwirtschaft
Forschungsbereich Digitaler Bauprozess
Technische Universität Wien
Karlsplatz 13/235-03, 1040 Wien, Österreich

Wien, im Oktober 2023 _____

Kurzfassung

Die modellbasierte Planung gewinnt im Bauwesen zunehmend an Bedeutung und steht im Zentrum der BIM-Methode, die darauf abzielt, umfassende Gebäudeinformationen in einem zentralen Modell zu integrieren. In diesem Kontext hat die Stadt Wien mit dem Forschungsprojekt BRISE-Vienna den Weg für eine zukünftige, openBIM-fähige Baueinreichung geebnet. Dies ermöglicht eine effizientere Behördenabwicklung bei der Anwendung der BIM-Methode. Obwohl heutige Gebäudemodelle bereits vielfältige Möglichkeiten zur Modellierung einzelner Bauteile bieten, fehlen einheitliche Standards für die Darstellung von Fluchtwegen, die einen entscheidenden Aspekt in Bezug auf die behördliche Überprüfung von Gebäuden darstellen.

Diese Masterarbeit beschäftigt sich mit der automatisierten Implementierung von Fluchtwegsinformationen in das standardisierte Dateiformat Industry Foundation Classes (IFC), das in der openBIM-Methode als grundlegendes Werkzeug für die Gebäudemodellierung dient. Die Arbeit basiert auf den Ergebnissen einer automatisierten Fluchtwegsanalyse und entwickelt fünf verschiedene Modellierungsansätze, um eine sinnvolle Integration von Fluchtwegen in das IFC-Schema zu erreichen. Dabei wird der genaue Aufbau und die zugrunde liegende IFC-Struktur der einzelnen Modelle untersucht. Die Umsetzung dieser Konzepte erfolgt mithilfe der Softwarebibliothek IfcOpenShell, die auf Python basiert, und bietet eine detaillierte Darstellung des Modellierungsprozesses.

Jeder Fluchtweg in dem generierten IFC-Modell besitzt drei grundlegende Eigenschaften: seinen Verlauf durch das Gebäude, eine Gesamtlänge und eine Personenanzahl. Diese Informationen sind für die spätere Gebäudenutzung von großer Bedeutung. Eine Archivierung dieser Daten im Rahmen des IFC-Formates stellt damit einen wertvolles Hilfsmittel für Behördenüberprüfungen und die Sicherheit des Bauwerkes dar. Die Ergebnisse der Modellierung zeigen, dass IfcOpenShell ein leistungsstarkes Werkzeug für die Verarbeitung von IFC-Dateien ist, jedoch ein umfangreiches Verständnis der IFC-Datenstruktur erfordert. Der Vergleich der fünf untersuchten Modelle zeigt, dass die Verwendung einer segmentweisen Aufteilung von Fluchtwegen die besten Ergebnisse liefert. Bei dieser Variante fassen eigene Gruppen die einzelnen Segmente in den Modellen wieder zusammen und rekonstruieren somit die gesamten Fluchtwegsverläufe. Diese Methode ermöglicht es, die Fluchtwegsverläufe in einem einzigen IFC-Modell umfassend darzustellen und zu speichern. Damit wird eine wichtige Grundlage für die effiziente Integration von Fluchtwegen in BIM-Modelle geschaffen. Dies trägt zu einer weiteren Optimierung des BIM-Modellierungsprozesses bei und fördert die Sicherheit und Effizienz in der Bauplanung.

Abstract

Model based planning is becoming increasingly important in the construction industry and is at the core of the BIM method, which aims to integrate comprehensive building information into a central model. In this context, the city of Vienna has paved the way for a future openBIM compatible building permit submission through the research project BRISE-Vienna. This enables more efficient government processing when applying the BIM method. While current building models offer various possibilities for modeling individual components, there are no uniform standards for representing escape routes, which are a crucial aspect of building regulatory inspections.

This master's thesis focuses on the automated implementation of escape route information into the standardized Industry Foundation Classes (IFC) file format, which serves as a fundamental tool for building modeling in the openBIM method. The work is based on the results of an automated escape route calculation and develops five different modeling approaches to achieve a meaningful integration of escape routes into the IFC schema. It examines the precise structure and underlying IFC structure of each model. The implementation of these concepts is done using the IfcOpenShell software library, which is based on Python, and provides a detailed representation of the modeling process.

Each escape route in the generated IFC model has three fundamental properties: its path through the building, a total length, and a number of people. These pieces of information are of great importance for the subsequent use of the building. Archiving this data within the IFC format thus represents a valuable tool for regulatory inspections and the safety of the structure. The modeling results show that IfcOpenShell is a powerful tool for processing IFC files but requires a comprehensive understanding of the IFC data structure. Comparing the five models investigated, it is evident that the use of a segmented division of escape routes yields the best results. In this variant, individual segments are grouped together in the models to reconstruct the entire escape route paths. This method allows for comprehensive representation and storage of escape route paths in a single IFC model, laying an important foundation for the efficient integration of escape routes into BIM models. This contributes to further optimizing the BIM modeling process and enhances safety and efficiency in construction planning.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Zuge stetig komplexerer Planungen von Gebäuden spielt die BIM-Methode (Building Information Modeling) eine immer tragendere Rolle [30]. Sie vereint alle klassischen Planungsdisziplinen miteinander. Dadurch kann ein BIM-Modell eine Vielzahl an Informationen über ein Gebäude aufnehmen und zur Verfügung stellen [3]. Dabei kann es sich um die Positionen und Dimensionen konstruktiver Bauteile, aber auch um virtuelle Elemente und Konzepte handeln. Zu Letzteren gehören die Angaben zu Fluchtwegen in einem Bauwerk. Fluchtwege sind ein sicherheitskritischer Aspekt für jedes Gebäude, da sie in einem Brandfall eine wichtige Rettungseinrichtung des baulichen Brandschutzes darstellen [23]. Daher muss auf die Planung und behördliche Überprüfung dieser ein besonderes Augenmerk gelegt werden. Ein BIM-Modell kann hierbei den Aufwand maßgeblich verringern, da es die Möglichkeit von automatisierten Überprüfungen bietet [12]. Besonders die behördliche Überprüfung kann durch dieses Hilfsmittel unterstützt werden [14]. Auch abseits dieser Prüfung sollten Informationen über einzelne Fluchtwege und deren Verläufe in einem BIM-Modell eines Gebäudes enthalten sein. Im Angesicht der openBIM-Methode soll das Gebäudemodell zusätzlich in einem offenen Datenformat aufgebaut sein. Einerseits dient das einer universellen Verwendbarkeit dieser Daten von unterschiedlichsten Personen, andererseits stellt es eine sichere Form der Archivierung dieser Informationen dar. Das hierfür am weitesten verbreitete und auch als ISO-Standard verifizierte Dateiformat ist das Format Industry Foundation Classes (IFC) [9]. Es bietet bereits einen großen Umfang an Möglichkeiten für die Modellierung einzelner Bestandteile eines Bauwerkes. Gleichzeitig ist der Facettenreichtum von Gebäuden praktisch grenzenlos. Daher kann es im Rahmen dieses Dateiformates nicht für alle gewünschten Objekte vorgefertigte Lösungen geben. Einer dieser Fälle sind Fluchtwege.

In dieser Arbeit liegt der Fokus auf der Integration von Fluchtwegsdaten in einer IFC-Datei, wobei zwei zentrale Forschungsfragen im Mittelpunkt stehen. Auf der konzeptionellen Ebene ist die erste Frage, wie Fluchtwegsinformationen effektiv innerhalb des IFC-Datenschemas abgebildet werden können. Hierbei ist von besonderem Interesse, wie Informationen zu individuellen Fluchtwegen in einer IFC-Datei gespeichert werden können, um ihre zukünftige Sicherheit zu gewährleisten. Eine Herausforderung besteht darin, dass das IFC-Schema kein dediziertes Element zur Darstellung von Fluchtwegen vorsieht. Daher stehen verschiedene Alternativen zur Implementierung solcher Elemente zur Verfügung, von dreidimensionalen Proxy-Objekten bis hin zu zweidimensionalen Annotationen. Eine Variantenstudie wird durchgeführt, um diese Modellierungsvarianten zu untersuchen und ihre jeweiligen Vor- und Nachteile zu vergleichen.

Auf technischer Ebene wird die zweite Forschungsfrage angegangen: Kann der Prozess der Speicherung von Fluchtwegsdaten in einer IFC-Datei effizient automatisiert werden? Schnabel [31] hat in seiner Arbeit die Möglichkeiten der, auf JavaScript (JS) basierenden Software, IFC.js untersucht. Für die vorliegende Masterarbeit wird jedoch die Programmiersprache Python verwendet, um die gewonnenen Erkenntnisse in das IFC-Format umzusetzen. Dies umfasst die Speicherung der Geometriedaten und Fluchtwegsverläufe in der IFC-Datei, wobei als Ausgangsdaten die Ergebnisse einer vorherigen Fluchtwegsberechnung dienen. Die Automatisierung der Generierung eines Fluchtwegsmodells aus diesen Informationen wird unter Einsatz der IfcOpenShell-Bibliothek

durchgeführt. Zusätzlich wird eine Bewertung des Funktionsumfangs und der Anwenderfreundlichkeit dieser Bibliothek vorgenommen.

Diese Arbeit zeigt zuerst einen kurzen Überblick über das IFC-Format und die Funktionsweise der verwendeten IfcOpenShell-Bibliothek. Ebenfalls erkläre ich in den Grundlagen die allgemeine Definition eines Fluchtweges, die automatisierte Fluchtwegsanalyse und wo mit dem Projekt BRISE-Vienna [34, 36] ein möglicher Anwendungspunkt für die Ergebnisse liegt. Nachfolgend beschäftige ich mich mit den verschiedenen Modellierungsvarianten der Fluchtwege, welche in dieser Arbeit untersucht werden. Hierbei gehe ich zuerst auf die konzeptionellen Ideen ein und beschreibe anschließend die tatsächliche Umsetzung im IFC-Format. Im darauffolgenden Teil betrachte ich die Implementierung der Konzepte mit der IfcOpenShell-Bibliothek. Das erfolgt an der detaillierten Erklärung einer Variante und wie das Programm daraus eine IFC-Datei generiert. Schlussendlich bewerte ich die Resultate dieser Arbeit. Hierbei möchte ich die betrachteten Modellierungsvarianten nach verschiedenen Aspekten bewerten. Abschließend folgt eine kurze Zusammenfassung und ein Ausblick.

Kapitel 2

Grundlagen

In diesem Kapitel beschreibe ich einige Aspekte des IFC-Datenformates und der Softwarebibliothek IfcOpenShell. Außerdem befasse ich mich mit der Definition eines Fluchtweges sowie dem Anwendungsfall der digitalen Baueinreichung.

2.1 IFC – Industry Foundation Classes

buildingSMART beschreibt das IFC-Format als deren primären, technischen Beitrag, um die openBIM-Methode als internationalen Standard zu etablieren [9]. Das IFC-Format ist hierbei ein zentraler Baustein in der Umsetzung einer openBIM-basierten Arbeitsweise. Es stellt ein herstellerneutrales und frei zugängliches Datenformat dar, welches alle Bauwerksinformation bündelt. Das Format wurde von buildingSMART entwickelt und ist in dem ISO-Standard ISO 16739 [18] veröffentlicht. Es basiert auf der Datenmodellierungssprache EXPRESS, welche im Teil 11 des STEP-Standards (ISO 10303-11) [17] geregelt ist [13].

Eichler et al. [13] beschreiben die zugrundeliegende Architektur des IFC-Schemas in allen Einzelheiten. Ebenso bietet die Dokumentation von buildingSMART [5] in seiner neuesten Version einen genauen Überblick des IFC-Datenformats und dessen Anwendung. Die nachfolgende Erläuterung einer IFC-Datei ist in Anlehnung an diese beiden Dokumente erstellt.

Eine IFC-Datei besteht in ihren Grundbausteinen aus einzelnen Objekten. Dabei entspricht jede Zeile einer IFC-Datei genau einem dieser Objekte. Jedes Objekt besitzt eine ExpressID, eine Klasse (engl.: Entity) und unterschiedlich viele Attribute. Abbildung 2.1 zeigt eine Zeile

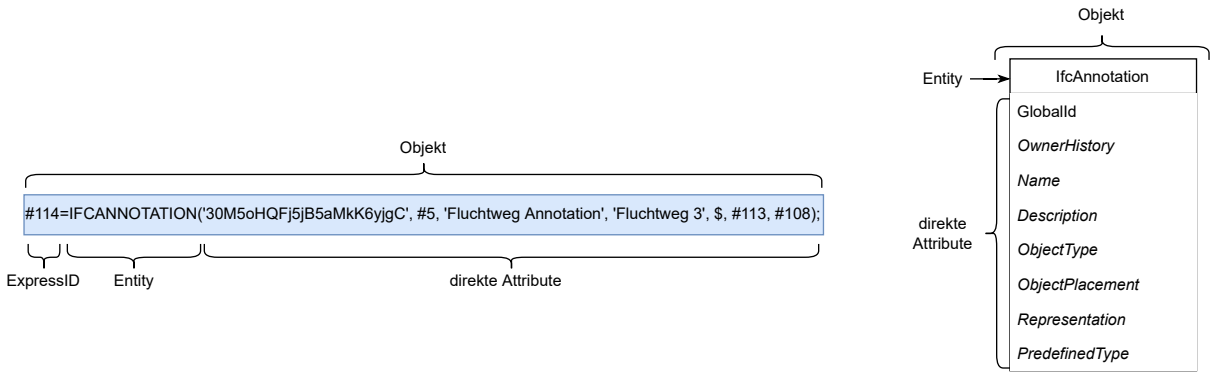


Abb. 2.1: Die linke Darstellung zeigt ein Objekt, wie es in einer IFC-Datei vorkommt. Die Attribute sind durch ein Komma getrennt. Das zweite Attribut #5 verweist auf ein anderes Objekt über dessen ExpressID. Das vierte Attribut 'Fluchtweg 3' besitzt diesen Namen als direkten Wert. An der fünften Stelle ist ein Nullwert als Dollarzeichen ($) eingetragen. Die rechte Darstellung eines Objektes wird für die weiteren Abbildungen in dieser Arbeit verwendet.

einer IFC-Datei und wie diese in den weiteren Grafiken dieser Arbeit dargestellt wird. Jedem Objekt in einer IFC-Datei wird eine individuelle ExpressID zugeteilt. Über diese können Objekte aufeinander verweisen und sich gegenseitig referenzieren. Die Entity definiert, was das Objekt im Rahmen des IFC-Schemas darstellt. Zum Beispiel ist eine Wand ein Objekt der Klasse *IfcWall*. Die Entity legt außerdem fest, welche Attribute ein Objekt besitzt. Attribute beschreiben das Objekt näher. Sie können entweder auf ein anderes Objekt über die ExpressID verweisen, einen definierten Wert oder einen Null-Wert ($) besitzen.

Das IFC-Datenschema arbeitet mit dem Prinzip der objektbasierten Vererbung. Das bedeutet, dass eine Entity eine Subklasse von anderen Entities darstellt. Eine Entity besitzt daher nicht nur ihre eigenen Attribute, sondern auch die der ihr übergeordneten Superklassen. Dabei wird jedoch nicht der Inhalt der Attribute vererbt, sondern nur die Attributstruktur selbst. Es kann für jede Entity eine Vererbungshierarchie wie in Abbildung 2.2 erstellt werden. Hierbei stellt die Superklasse *IfcRoot* immer den Ausgangspunkt dar. Die meisten der vererbten Attribute werden nicht für jedes Objekt angegeben. Einerseits werden viele Attribute aus Beziehungen mit anderen Objekten bestimmt, andererseits sind manche Attribute optional und müssen nicht befüllt werden. buildingSMART gibt hierbei die genaue Verwendung der Attribute vor, wie in Abbildung 2.3.

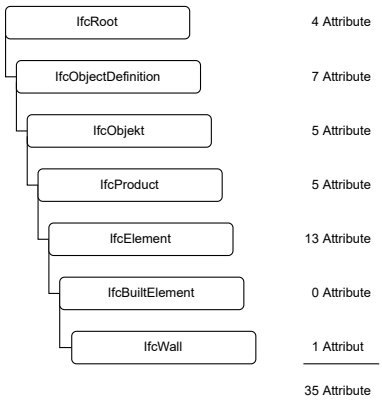


Abb. 2.2: Vererbungshierarchie der Entity *IfcWall* mit der jeweiligen Anzahl an Attributen, welche aus jeder Ebene vererbt werden.

In der IFC-Datenschema-Architektur existieren vier konzeptionelle Layer. Diese sind der *Core Layer*, *Interoperability Layer*, *Domain Layer* und der *Resource Layer*. Eine genaue Beschreibung jedes Layers geben Eichler et al. [13], sie werden an dieser Stelle nicht näher erläutert. Die Layer gruppieren alle Entities des IFC-Schemas. Der *Core Layer*, *Interoperability Layer* und *Domain Layer* enthalten gemeinsam alle Entities, welche einen Global Unique Identifier (GUID) besitzen und als eigenständige Objekte existieren können. Der *Resource Layer* ist separat von den restlichen Layern zu betrachten. Die Entities dieses Bereiches können nicht alleine existieren, sondern müssen von Objekten aus anderen Layern referenziert werden. Hier sind Objekte enthalten wie Merkmale, geometrische Punkte und Linien sowie Entities zur Platzierung von Objekten. Abbildungen 2.4 und 2.5 zeigen alle Entities und deren Zugehörigkeit, welche in dieser Arbeit verwendet werden.

Die nachfolgende Übersicht bezieht sich auf die Erläuterungen von Eichler et al. [13] und auf Übersetzungen der IFC4.3-Spezifikation von buildingSMART [10]. Die Liste führt nur Begriffe des IFC-Schemas an, welche im Zuge dieser Arbeit verwendet werden.

#	Attribute	Type	Description
IfcRoot (4)			
1	GlobalId	IfcGloballyUniqueId	Assignment of a globally unique identifier within the entire software world.
2	OwnerHistory	OPTIONAL IfcOwnerHistory	Assignment of the information about the current ownership of that object, including owning actor, application, local identification and information captured about the recent changes of the object. NOTE Only the last modification is stored - either as addition, deletion or modification. IFC4-CHANGE The attribute has been changed to be OPTIONAL.
3	Name	OPTIONAL IfcLabel	Optional name for use by the participating software systems or users. For some subtypes of IfcRoot the insertion of the Name attribute may be required. This would be enforced by a where rule.
4	Description	OPTIONAL IfcText	Optional description, provided for exchanging informative comments.
IfcObjectDefinition (7)			
	HasAssignments	SET [0:?] OF IfcRelAssigns FOR RelatedObjects	Reference to the relationship objects, that assign (by an association relationship) other subtypes of *IfcObject* to this object instance. Examples are the association to products, processes, controls, resources or groups.
	Nests	SET [0:1] OF IfcRelNests FOR RelatedObjects	References to the decomposition relationship being a nesting. It determines that this object definition is a part within an ordered whole/part decomposition relationship. An object occurrence or type can only be part of a single decomposition (to allow hierarchical structures only). IFC4-CHANGE The inverse attribute datatype has been added and separated from *Decomposes* defined at *IfcObjectDefinition*.

Abb. 2.3: Ausschnitt aus der Auflistung der Attribute der Entity *IfcWall* aus der Dokumentation von buildingSMART [5]. Das erste Attribut *GlobalId* hat eine Positionsnummer und ist damit ein direktes Attribut, welches zwingend erforderlich ist. Die *OwnerHistory* ist ebenfalls ein direktes Attribut, muss jedoch nicht ausgefüllt werden, da sie optional ist. Die Attribute *HasAssignments* und *Nests* sind beides keine direkten Attribute und werden nicht bei den Objekten bestimmt. Sie ergeben sich aus der restlichen IFC-Datei, falls sie benötigt werden.

Klasse (Engl.: Entity)

buildingSMART [10] definiert eine Entity nach der ISO 10303-11 [17] als eine Gruppierung von Informationen, welche durch gemeinsame Merkmale beschrieben ist. Jedes Element einer IFC-Datei gehört zu einer Entity. Sie kann als eine Schablone betrachtet werden, welche das Grundgerüst für ein Objekt festlegt. Die Attributsstrukturen werden zwischen den Entities über Vererbung weitergegeben.

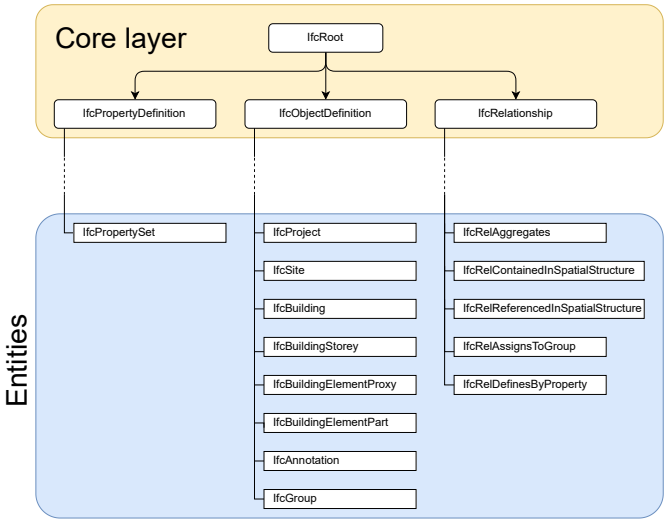


Abb. 2.4: Verwendete Entities aus dem *Core Layer* der IFC-Datenstruktur. Diese Darstellung ist eine Vereinfachung und zeigt nicht die tatsächliche Vererbungsstruktur der Entities mit allen Zwischenschritten.

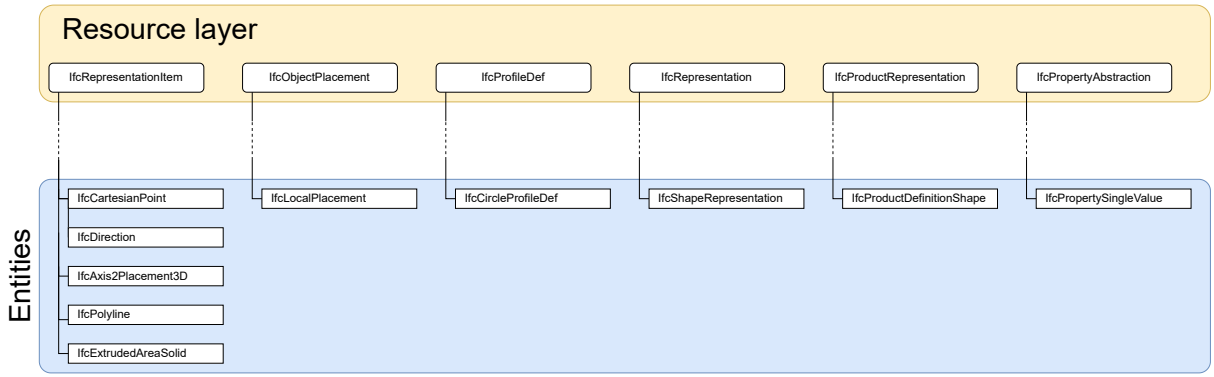


Abb. 2.5: Verwendete Entities aus dem *Resource Layer* der IFC-Datenstruktur. Diese Darstellung ist eine Vereinfachung und zeigt nicht die tatsächliche Vererbungsstruktur der Entities mit allen Zwischenschritten.

Objekt (Engl.: Object)

Ein Objekt ist eine Instanz einer Entity. Es stellt einen greifbaren Gegenstand oder ein vorstellbares Konzept dar, wie zum Beispiel ein Fenster oder einen Raum. Das Objekt kann als die Verwendung einer Entity betrachtet werden. Jede Zeile in einer IFC-Datei ist ein eigenständiges Objekt.

Attribute (Engl.: Attribute)

Attribute sind die notwendigen Merkmale, Qualitäten und Eigenschaften einer Entity. Jedes Objekt besitzt eine Fülle an verschiedenen Attributen aus der Vererbungsstruktur. Sobald eines dieser verwendet bzw. befüllt wird, findet eine Unterteilung in direkte, inverse und abgeleitete Attribute statt. Direkte Attribute erhält jedes Objekt individuell bei seiner Erstellung. Diese können je nach Entity des Objektes erforderlich oder optional sein. Inverse Attribute werden aus Entities abgeleitet, welche das Objekt referenzieren – zum Beispiel die Information, ob ein Objekt Teil einer Gruppe ist. Abgeleitete Attribute ergeben sich aus einer Berechnung, sobald dies möglich ist. So kann zum Beispiel die Oberfläche eines Bauteils ermittelt werden, sobald diesem eine Geometrie zugeteilt wurde.

Merkmal (Engl.: Property)

Ein Merkmal ist eine Eigenschaft, welche einem Objekt abseits seiner Attribute zugeordnet wird. Hierbei kann es sich um jede Art von Information handeln. Ein Merkmal wird als ein eigenes Objekt abgespeichert. Die Verknüpfung zu einem Element erfolgt über eine eigene Entity, welche eine Subklasse von *IfcRelationship* ist. Merkmale ermöglichen es, jedes Objekt mit Informationen zu bestücken, für die keine Attribute zur Verfügung stehen.

Merkmalliste (Engl.: Property Set/ PSet)

Ein PSet ist eine Zusammenfassung einzelner Merkmale zu einer Gruppe. Es kann auch nur aus einem einzelnen Property bestehen. Dadurch können mehrere Merkmale gemeinsam einem Objekt zugeordnet werden. Das PSet ist immer notwendig, um Merkmale mit Objekten zu verknüpfen. Es stellt die Brücke zwischen den beiden dar.

2.2 Grundlagen zur Anwendung von IfcOpenShell

IfcOpenShell ist eine auf C++ basierende Softwarebibliothek, welche eine Interaktion mit Dateien des IFC-Formates ermöglicht [16]. Das Paket ist frei verfügbar unter der Lesser General Puplic

License Version 3 (LGPL-3.0) [27]. IfcOpenShell ist mit den Programmiersprachen Python und C++ verwendbar. Die Bibliothek ermöglicht die Auswertung, Bearbeitung und Generierung von IFC-Dateien. Das Toolkit unterstützt die Versionen IFC2x3 und IFC4 Add2 TC1 in vollem Umfang, sowie bereits die Auswertung und Bearbeitung für IFC4.3. Die Kernfunktionen der Bibliothek umfassen das Auslesen der Daten einer IFC-Datei, Erfassen von direkten und inversen Attributen, Generieren, Ändern und Löschen von Entities und Erstellen neuer GUIDs [16].

Öffnen und Erstellen einer IFC-Datei

Um eine Bearbeitung an einem Modell durchzuführen, muss eine IFC-Datei geladen werden. Der Befehl `ifcopenshell.open(...)` öffnet eine existierende Datei. Eine komplett neue IFC-Datei erstellt der Befehl `ifcopenshell.file(...)`. Hierbei kann als Attribut das gewünschte IFC-Schema übergeben werden. Die Umsetzung zeigt Programmcode 2.1. Diese beiden Methoden gehören zu den bibliothekspezifischen Befehlen und erhalten daher den Präfix `ifcopenshell`. Um den tatsächlichen Inhalt einer Datei zu bearbeiten werden Methoden immer direkt auf diese angewendet. Alle abgebildeten Codesegmente verwenden hierfür die Variable `ifcfile`.

```
# Öffnen einer IFC-Datei unter dem angegebenen Pfad
ifcfile = ifcopenshell.open('/path/model.ifc')

# Erstellen einer neuen IFC-Datei
ifcfile = ifcopenshell.file()
ifcfile = ifcopenshell.file(schema='IFC4')
```

Programmcode 2.1: Öffnen und Erstellen einer IFC-Datei

Auslesen einer IFC-Datei

Mit verschiedenen Funktionen kann die zuvor eingelesene IFC-Datei ausgelesen werden (siehe Programmcode 2.2). Der Befehl `.by_id(...)` kann direkt das Objekt der IFC-Datei mit der gewünschten ExpressID ausgeben. Auch eine direkte Suche nach einer GUID ist mit `.by_guid(...)` möglich. Der Befehl `.by_type(...)` durchsucht die Datei nach allen Objekten einer bestimmten Entity.

```
# Gibt das Objekt mit der ExpressID 3 der Datei wieder
objekt = ifcfile.by_id(3)

# Gibt das Objekt mit der gesuchten GUID wieder
objekt = ifcfile.by_guid('2ammZWO9nBBwhA1ZZ$gSVp')

# Gibt alle Objekte der Entity IfcWall als Liste wieder
walls = ifcfile.by_type('IfcWall')
wall = walls[1]
```

Programmcode 2.2: Auslesen von Objekten aus einer IFC-Datei

Erfassen von Attributen

Einzelne Attribute kann IfcOpenShell direkt aus der Instanz einer Entity extrahieren. Die Bibliothek speichert dabei die einzelnen Entities als Python-Objekte ab. Diese Objekte besitzen im Python-Skript die selben Attribute wie die Entities per Definition im IFC-Format. Damit folgt die Benennung exakt den Vorgaben des IFC-Datenschemas. Programmcode 2.3 zeigt einige dieser Anwendungen. Beispielsweise kann die GUID eines Objektes mit dem Befehl `.GlobalId`

direkt abgerufen werden. Der Befehl `.get_inverse(...)` gibt eine Liste mit allen Einträgen der IFC-Datei aus, welche das Objekt referenzieren. Im Gegensatz dazu kann mit dem Befehl `.traverse(...)` jedes Element gezeigt werden, welches von dem Objekt referenziert wird.

```
# Gibt die GUID der Entity wieder
wall.GlobalId

# Gibt eine Liste der referenzierenden Entitys wieder
ifcfile.get_inverse(wall)

# Gibt eine Liste der referenzierten Entitys wieder
ifcfile.traverse(wall, max_levels=1)
```

Programmcode 2.3: Auslesen von unterschiedlichen Attributen aus einer IFC-Datei

Erstellen von GUIDs

Die GUID ist für jede Entity einer IFC-Datei individuell. Programmcode 2.4 zeigt, wie eine neue GUID mit dem bibliothekspezifischen Befehl `ifcopenshell.guid.new()` erstellt werden kann.

```
# Erstellen einer neue GUID
GUID = ifcopenshell.guid.new()
```

Programmcode 2.4: Erstellen einer neuen GUID

Bearbeiten von Entities

Im Zentrum der Erstellung neuer Einträge in einer IFC-Datei steht der Befehl `create`. Dieser wird im Programmcode 2.5 gezeigt. Um eine bestimmte Entity zu erstellen, kann diese entweder direkt an den `create` Befehl angehängt oder als Attribut an den `.create_entity(...)` Befehl übergeben werden. Die restliche Attribute des Befehls entsprechen den direkten Attribute nach der Definition des IFC-Schemas. Über die Befehlsform `.createIfcWall(...)` kann beispielsweise eine *IfcWall* in der Datei erstellt werden. Jeder `create` Befehl generiert einen neuen Eintrag in der IFC-Datei an der nächsten, freien ExpressID. Die auf das `create` folgende Benennung bestimmt die zu erstellende Entity, zum Beispiel eine *IfcWall*. Es können alle im IFC-Schema definierten Entities [4] erzeugt werden. Die übergebenen Attribute des Befehls sind in der Reihenfolge und Anwendung identisch zu denen des IFC-Schemas der verwendeten Entity. Beispielsweise hat eine *IfcWall* laut der IFC-Dokumentation [7] 35 Attribute, von denen 9 festgelegt werden müssen. Bei den restlichen 26 handelt es sich um inverse oder abgeleitete Attribute. Die direkten Attribute können auch mit Schlüsselwörtern übergeben werden. Der Befehl `.remove(...)` löscht eine bestimmte Entity aus der IFC-Datei.

```
# zwei Varianten um eine IfcWall zu erstellen
ifcfile.createIfcWall()
ifcfile.create_entity('IfcWall')

# Erstellen einer IfcWall mit Attributen
ifcfile.createIfcWall(GUID, None, 'New Wall', None, None, None, None, None,
                      None)
ifcfile.create_entity('IfcWall', GUID, None, 'New Wall', None, None, None,
                      None, None, None)

# Leere Attribute müssen am Ende nicht übergeben werden
```

```
12 ifcfile.createIfcWall(GUID, None, 'New Wall')
13
14 # Attribute können als Schlüsselwort übergeben werden
15 ifcfile.createIfcWall(GlobalId=GUID, Name='New Wall')
16
17 # Löschen einer Entity aus der Datei
18 ifcfile.remove(wall)
```

Programmcode 2.5: Bearbeitung von Entities in einer IFC-Datei

2.3 Definition eines Fluchtwegs

In den Begriffsbestimmungen der OIB-Richtlinien [26, S. 5] ist ein Fluchtweg folgendermaßen definiert:

> *Weg, der den Benützern eines Bauwerkes im Gefahrenfall grundsätzlich ohne fremde Hilfe das Erreichen eines sicheren Ortes des angrenzenden Geländes im Freien – in der Regel eine Verkehrsfläche – ermöglicht.*

Die OIB-Richtlinie 2 für Brandschutz [24] legt die gesetzlichen Rahmenbedingungen für die Ausführung von Fluchtwegen fest. In dieser sind unter anderem die maximalen Längen und erforderlichen Breiten für verschiedene Gebäudeklassen festgelegt. Technisch gesehen kann ein Fluchtweg als ein dreidimensionaler Weg durch ein Gebäude betrachtet werden, welcher bestimmte geometrische Anforderungen erfüllen muss. Dieser Pfad ist in seiner einfachsten Form eine simple Abfolge von geraden Abschnitten. Zusätzlich zu dieser geometrischen Definition besitzt ein Fluchtweg eine maximale Personenanzahl, welche diesen benutzen können. Aus dieser lässt sich der notwendige Platzbedarf ermitteln. Konkret gibt die OIB-Richtlinie 4 [25] hier die nötigen Breiten von Wegen, Treppen, Türen und Rampen an. Zusätzlich sind in dieser Richtlinie noch weitere Aspekte festgelegt, wie z.B. Durchgangshöhen, Rampenneigungen. Die Angabe über die Anzahl der flüchtenden Personen stellt damit die Ausgangssituation eines Fluchtweges dar und soll im IFC-Datenformat zukunftssicher abgespeichert werden. In einem Gebäude ergeben sich aus der Nutzung eine Fülle von unterschiedlichen Fluchtwegsmöglichkeiten. Jeder betretbare Raum stellt einen möglichen Ausgangspunkt für eine flüchtende Person und damit einen eigenen Fluchtweg dar. So kann im Fall eines Brandes aus jedem Zimmer eine bestimmte Anzahl an Personen flüchten. Da nicht jeder Raum seinen individuellen Fluchtweg an einen sicheren Ort besitzen kann, bündeln öffentliche Verkehrsflächen wie Gänge und Treppenhäuser mehrere Fluchtwege zusammen. Aus vielen einzelnen Personen wird ein gemeinsamer Personenstrom geschaffen.

2.4 Fluchtwegsberechnung

Die automatische Berechnung von Wegen durch ein Gebäude kann unterschiedliche Anwendungsfälle finden. Lee et al. [22] messen mit einer computergestützten Lösung verschiedene Weglängen in Gebäudemodellen. Kannala [19] zeigte, dass 3D-Modelle ausreichende Informationen beinhalten, um Fluchtwege nach üblichen rechtlichen Voraussetzungen zu generieren und zu prüfen. Für diese Arbeit wird die Publikation von Fischer et al. [14] als Ausgangspunkt verwendet. In dieser wird eine automatisierte Fluchtwegsberechnung im Zuge des Forschungsprojektes BRISE-Vienna erarbeitet. Hierbei wurde ein gezielter Fokus auf die Einhaltung der rechtlichen Grundlagen in Österreich gesetzt. Damit kann der entwickelte Prozess für eine behördliche Prüfung angewendet werden.

Fluchtwege stellen spezielle Wege innerhalb eines Gebäudes dar. Um sie zu finden, müssen Aspekte wie der Startpunkt in einem Raum, der genaue Verlauf durch das Gebäude und die verschiedenen Brandabschnitte innerhalb des Bauwerkes berücksichtigt werden. Außerdem müssen die Fluchtwegsbreiten in Abhängigkeit von der Personenanzahl ausreichend sein. Fischer et al. [14] verwenden die drei nachfolgenden Schritte, um einen Fluchtweg zu generieren:

- Ein Navigationsmodell für den Innenraum des Gebäudes generieren,
- mit dem Dijkstra-Algorithmus die gesuchten Routen finden und
- alle Abschnitte entlang des Weges auf die nötigen Dimensionen prüfen.

Für das Navigationsmodell werden in allen Räumen Knotenpunkte definiert. Diese stellen die möglichen Verbindungspunkte für verschiedene Wege durch den Raum dar. Verbindungen zwischen den Räumen erfolgen über Türen, welche auch mit speziellen Knoten dargestellt werden. Für einen Raum wird der weitest entfernte Knoten zur Tür als Ausgangspunkt des Fluchtweges verwendet. Mit dem Dijkstra-Algorithmus wird dieser Knoten für jeden Raum ermittelt. In einer zweiten Berechnung werden die einzelnen Wege durch die Räume kombiniert, um eine gesamte Route durch das Gebäude zu erhalten. Im letzten Schritt werden alle Objekte entlang des Fluchtweges überprüft. Hierbei müssen zum Beispiel Türen und Treppen die notwendigen Breiten für die flüchtenden Personenzahlen vorweisen.

Die mit diesem Prozess gefundenen Fluchtwege bilden die Ausgangsdaten für die vorliegende Arbeit. Die von Fischer et al. [14] entwickelte Fluchtwegsberechnung ist in der Prüfsoftware Solibri Office [33] realisiert. Die Ergebnisse stehen in visueller Form nur in diesem Programm zur Verfügung. Eine Sicherung der gefundenen Fluchtwege im Rahmen des IFC-Formates stellt den Grundgedanken dieser Arbeit dar. Die Ergebnisse der Fluchtwegsberechnung werden hierfür als eine JSON-Datei abgespeichert. Diese enthält die wichtigsten Gebäudedaten, wie den Bauplatz und die einzelnen Stockwerke. Jeder Fluchtweg ist als eine Abfolge von 3D-Koordinatenpunkten angegeben, welche den Knoten aus dem Navigationsmodell der Berechnung entsprechen.

2.5 Digitale Baueinreichung – Projekt BRISE-Vienna

Seit 2019 ist in Wien die digitale Baueinreichung möglich. Bauverfahren müssen seither nicht mehr zwingend in Papierform an die Behörden übermittelt werden, sondern können digital eingereicht werden. Daraus ergeben sich Zeitersparnisse bei der Abwicklung des gesamten Bauprozesses. Den nächsten Schritt in diesem Antragssystem stellt die Verwendung von digitalen Bauwerksmodellen dar. Das soll ein openBIM-fähiges Bewilligungsverfahren auf Basis von 3D-Gebäudemodellen ermöglichen. Krischmann et al. [21] schreibt, dass die Vorteile eines BIM-Modells im Baubewilligungsverfahren noch weitestgehend ungenützt sind. Die Informationen eines solchen Modelles sollten auch der Baubehörde bei ihrer Überprüfung zur Verfügung stehen. Um das zu ermöglichen hat die Stadt Wien das Forschungsprojekt BRISE-Vienna (Buildings Regulations Information for Submission Evaluation) ins Leben gerufen [34, 36]. In diesem neuen Einreichungsprozess sollen Modelle teil-automatisiert bewertet werden. Ein Teilbereich dieses Projektes umfasste die Entwicklung einer automatischen Prüfroutine für Fluchtwege innerhalb eines Bauwerkes [14], welche die Grundlage für diese Arbeit darstellt.

Ein wichtiger Eckpunkt des Forschungsprojektes BRISE-Vienna ist die zwingende Erfordernis der openBIM-Methode [21]. Nur diese kann garantieren, dass keine Benachteiligungen unterschiedlicher Planungsbüros aufgrund der verwendeten Programme entstehen. Damit ist auch das IFC-Format als frei zugänglicher Dateistandard ein zentraler Baustein. Informationen, welche in einem zukünftig immer öfter verwendeten BIM-Modell hinterlegt sind, können baubedingte

Prozesse wie eine Einreichung und Bewilligung maßgebend beschleunigen. Eine prüfende Behörde profitiert ebenfalls maßgebend von diesem Prozess. Freigegebene Unterlagen, wie zum Beispiel eine Fluchtwegsplanung, können in dem Modell archiviert werden. Damit kann im Fall später auftretender, rechtlicher Fragestellungen auf einen qualitativ hochwertigen Planungsstand zurückgegriffen werden. Auch nach der Fertigstellung sind diese Informationen für den gesamten Lebenszyklus eines Gebäudes sinnvoll. Besonders für spätere Nutzungsänderungen und Umbauten stellt das BIM-Modell eine kostbare Hilfe dar [2]. Wenn Fluchtwegsdaten ebenfalls integriert werden, sind auch sie für den späteren Lebenszyklus eines Gebäudes abrufbar. Damit kann ein openBIM-Modell diese Gebäudeinformation ebenfalls zukunftssicher aufbewahren.

Kapitel 3

Darstellung von Fluchtwegen im IFC-Format

Im Rahmen des IFC-Formates gibt es keine standardisierte Variante, wie ein Fluchtweg darzustellen ist. In diesem Kapitel werden unterschiedliche Modellierungsmethoden für Fluchtwege beschrieben und danach im IFC-Datenschema aufgebaut. Hierbei benenne ich alle Entities einer IFC-Datei, welche für die einzelnen Varianten nötig sind. Die Verknüpfungen dieser werden mit Strukturdiagrammen erklärt.

3.1 Konzepte der Modellierung

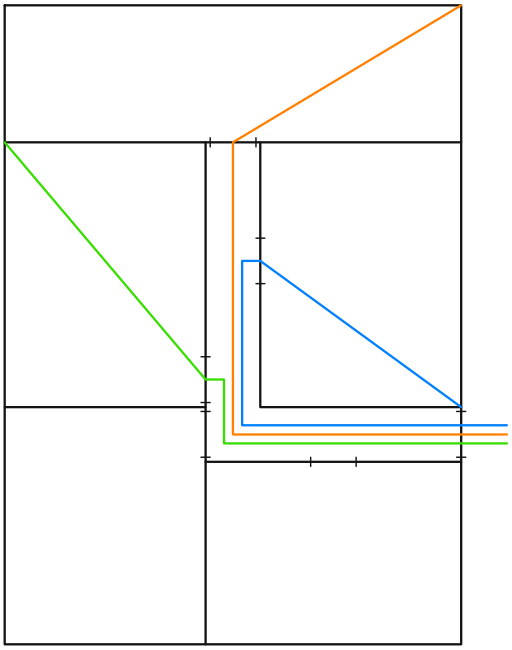
Jeder Fluchtweg in dem generierten IFC-Modell besitzt drei grundlegende Eigenschaften: seinen Verlauf durch das Gebäude, eine Gesamtlänge und eine Personenanzahl. Je komplexer ein Gebäude ist, desto anspruchsvoller werden auch die Fluchtwegskonzepte. Mit zunehmender Nutzfläche und Raumanzahl steigt die Menge der möglichen und auch geforderten Fluchtwege an. Bereiche, wie Treppenhäuser und Gänge, werden unweigerlich von mehreren dieser Wege gemeinsam benutzt. Im Gefahrenfall müssen diese Abschnitte den Anforderungen an die dann auftretenden, größeren Personenanzahlen genügen. Da im IFC-Format jedem Fluchtweg physische Elemente zu Grunde liegen, kann es in einem 3D-Modell zu Kollisionen dieser Fluchtwegsobjekte kommen. Trotz dieser Überlappungen sollen alle drei Grundinformationen der einzelnen Fluchtwege in der IFC-Datei erhalten bleiben. Es bieten sich daher mehrere Konzepte für die Modellierung an.

3.1.1 Gesamter Fluchtweg als ein Element

Jeder Fluchtweg wird als ein gesamtes Element von Anfangs- bis Endpunkt definiert, siehe Abbildung 3.1. Diese einfache Modellierung übernimmt die Eingangsdaten praktisch direkt. Der gesamte Fluchtweg besteht aus einem Element, welches Informationen über die Personenanzahl und Länge des Fluchtweges enthält. Diese Variante erzeugt jedoch überlappende Fluchtwege. Hierbei ist die gesamte Personenanzahl je Abschnitt nicht ersichtlich, da die einzelnen Fluchtwegsabschnitte nicht addiert werden.

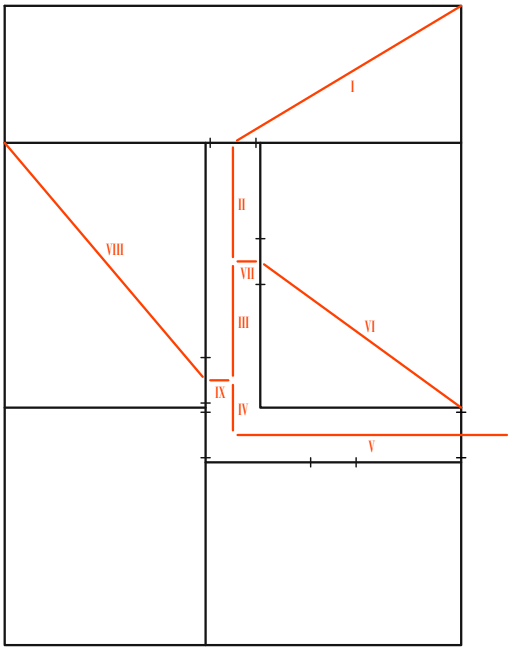
3.1.2 Fluchtwege als einzelne Segmente

Bei dieser Modellierung stellen einzelne Segmente gemeinsam einen ganzen Fluchtweg dar, wie in Abbildung 3.2. Die Eingangsdaten werden als individuelle Abschnitte generiert. Bei einer Überlappung mehrerer Fluchtwege werden die entsprechenden Segmente nur einmal erstellt. Hierbei erhöht sich die Personenanzahl der Abschnitte, um den gesamten Personenstrom abzubilden. Die Segmente können zusätzlich über ein separates Merkmal den Fluchtwegen zugeordnet werden. Die einzelnen Fluchtwegsverläufe sind damit wieder rekonstruierbar. Die Angaben zur Gesamtlänge der einzelnen Fluchtwege gehen jedoch verloren und können später nur aus der Geometrie der erstellten Segmente ermittelt werden.



Fluchtweg Nr.	Personenanzahl	Länge
1. Fluchtweg	2 Personen	16m
2. Fluchtweg	3 Personen	14m
3. Fluchtweg	5 Personen	12m

Abb. 3.1: Modellierungsvariante mit Fluchtwegen als ein gesamtes Element

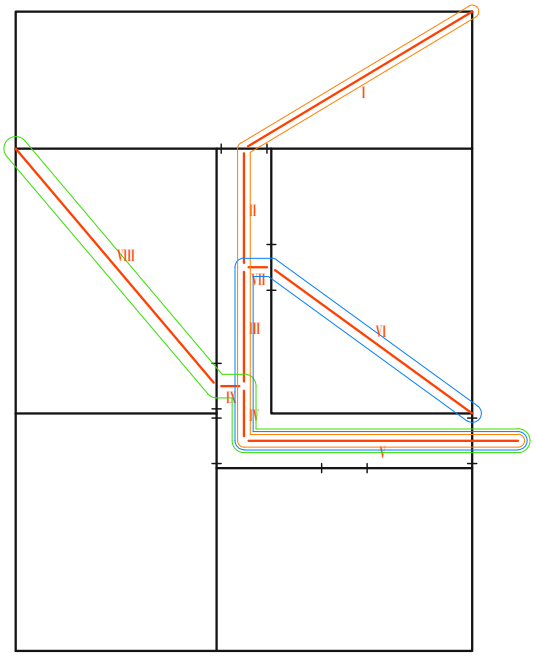


Segment	Personenanzahl	Zugehörigkeit		
I	2 Personen	1		
II	2 Personen	1		
III	5 Personen	1	2	
IV	10 Personen	1	2	
V	10 Personen	1	2	3
VI	3 Personen		2	
VII	3 Personen		2	
VIII	5 Personen			3
IX	5 Personen			3

Abb. 3.2: Modellierungsvariante mit Fluchtwegen aus einzelnen Segmenten, Zuordnung dieser über Merkmale

3.1.3 Fluchtwege als Gruppierung mehrerer Segmente

Wie zuvor zerteilt diese Modellierung die Fluchtwege wieder in einzelne Segmente. Diese erhalten jedoch keine Informationen über ihre Zugehörigkeiten. Stattdessen gruppieren zusätzliche Objekte die Segmente wieder zu einem gesamten Fluchtweg, wie in Abbildung 3.3 dargestellt. Dadurch ergeben sich neue Gruppenelemente, welche den Fluchtwegsverlauf darstellen. Diese Objekte bekommen als Merkmal die Gesamtlänge des Fluchtweges angefügt. Die Personenanzahlen werden addiert und als kumulative Wert den einzelnen Segmenten zugeordnet. Somit gibt es auch hier für jeden Abschnitt nur ein einziges Segment ohne Überlagerungen.



Fluchtweg Nr.	Länge
1. Fluchtweg	16m
2. Fluchtweg	14m
3. Fluchtweg	12m

Segment	Personenanzahl
I	2 Personen
II	2 Personen
III	5 Personen
IV	10 Personen
V	10 Personen
VI	3 Personen
VII	3 Personen
VIII	5 Personen
IX	5 Personen

Abb. 3.3: Modellierungsvariante mit Fluchtwegen aus einzelnen Segmenten, Zuordnung dieser über Zusammenfassung zu einer Gruppe

3.2 Aufbau einer IFC-Datei für Fluchtwege

Die zuvor beschriebenen Modellierungsvarianten können alle auf unterschiedliche Weise im IFC-Format realisiert werden. Zwischen den Varianten ändern sich lediglich die Entities, welche die eigentlichen Fluchtwegsobjekte darstellen. Ein Großteil der verwendeten Entities dient nicht direkt der Darstellung dieser Objekte. Die globalen Projektparameter sowie die Verortungsstruktur stellen das Grundgerüst dar, in welches die Fluchtwege eingefügt werden. Diese sind somit der eigentlichen Modellierung vorangestellt. Abbildung 3.4 zeigt den gesamten Dateibaum dieses Grundgerüstes. Die verwendeten Entities sind in vier Bereiche eingeteilt: *Header*, *Bauwerkshierarchie*, *Positionierung der Bauwerkshierarchie* und *Verortung*.

3.2.1 Header

Der Header-Bereich besitzt verschiedene Entities, siehe Abbildung 3.5. Die Darstellung orientiert sich an der IFC-Dokumentation [4]. Jedes Objekt ist mit seinen direkten Attributen und seiner Entity dargestellt. Kursiv dargestellte Attribute sind optional. In der fertigen IFC-Datei

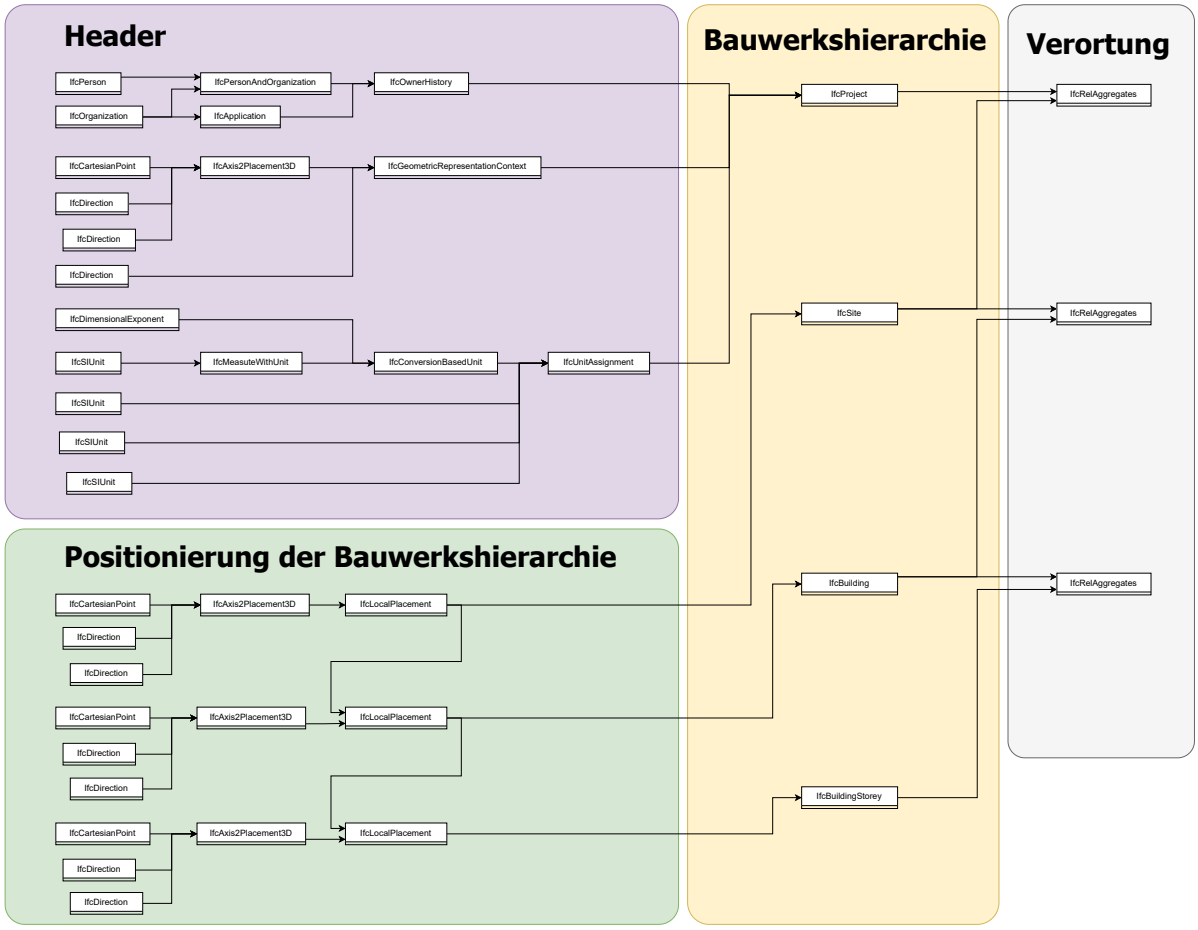


Abb. 3.4: Strukturdiagramm der zugrundeliegenden IFC-Datei für ein Fluchtwegsmodell

entspricht jede Zeile genau einem Baustein der Diagramme. Die *IfcOwnerHistory* beinhaltet Informationen über den Autor sowie die verwendete Software zur Erstellung der IFC-Datei. *IfcGeometricRepresentationContext* definiert die globalen Koordinatenrichtungen sowie die Präzision numerischer Werte, mit denen das Modell arbeitet. Zuletzt wird mit *IfcUnitAssignment* das verwendete Einheitensystem festgelegt. Für dieses Projekt verwendet die IFC-Datei die Einheiten m, m^2, m^3 sowie rad als *IfcSIUnit*. Die Einheit rad wird jedoch nur für einen Umrechnungsfaktor benötigt. Damit arbeiten alle Winkel in dem Modell in Grad. Die Angaben zum Befüllen der Entities des Header-Bereiches werden für das Modell abseits der eigentlichen Erstellung der Fluchtwege festgelegt.

3.2.2 Positionierung der Bauwerkshierarchie

Dieser Bereich, dargestellt in Abbildung 3.6, definiert die physikalische Positionierung der Bauwerksstruktur im Raum. Jeder Entity wird hier ein eigenes Koordinatensystem zugewiesen. Die Entity *IfcAxis2Placement3D* verbindet einen Nullpunkt in Form eines *IfcCartesianPoint* mit einem Achsensystem, welches durch zwei *IfcDirection* definiert ist. Damit ist ein vollständiger Einfügepunkt erstellt. Die Entity *IfcLocalPlacement* ermöglicht die Verbindung zu einem Element der Bauwerkshierarchie. Gleichzeitig kann bestimmt werden, ob sich der Einfügepunkt auf den globalen Nullpunkt oder ein anderes *IfcLocalPlacement* bezieht. Für jede Entity der IFC-Datei, welche ein Objekt im Raum darstellt, muss ein *IfcLocalPlacement* (Einfügepunkt) erstellt werden.

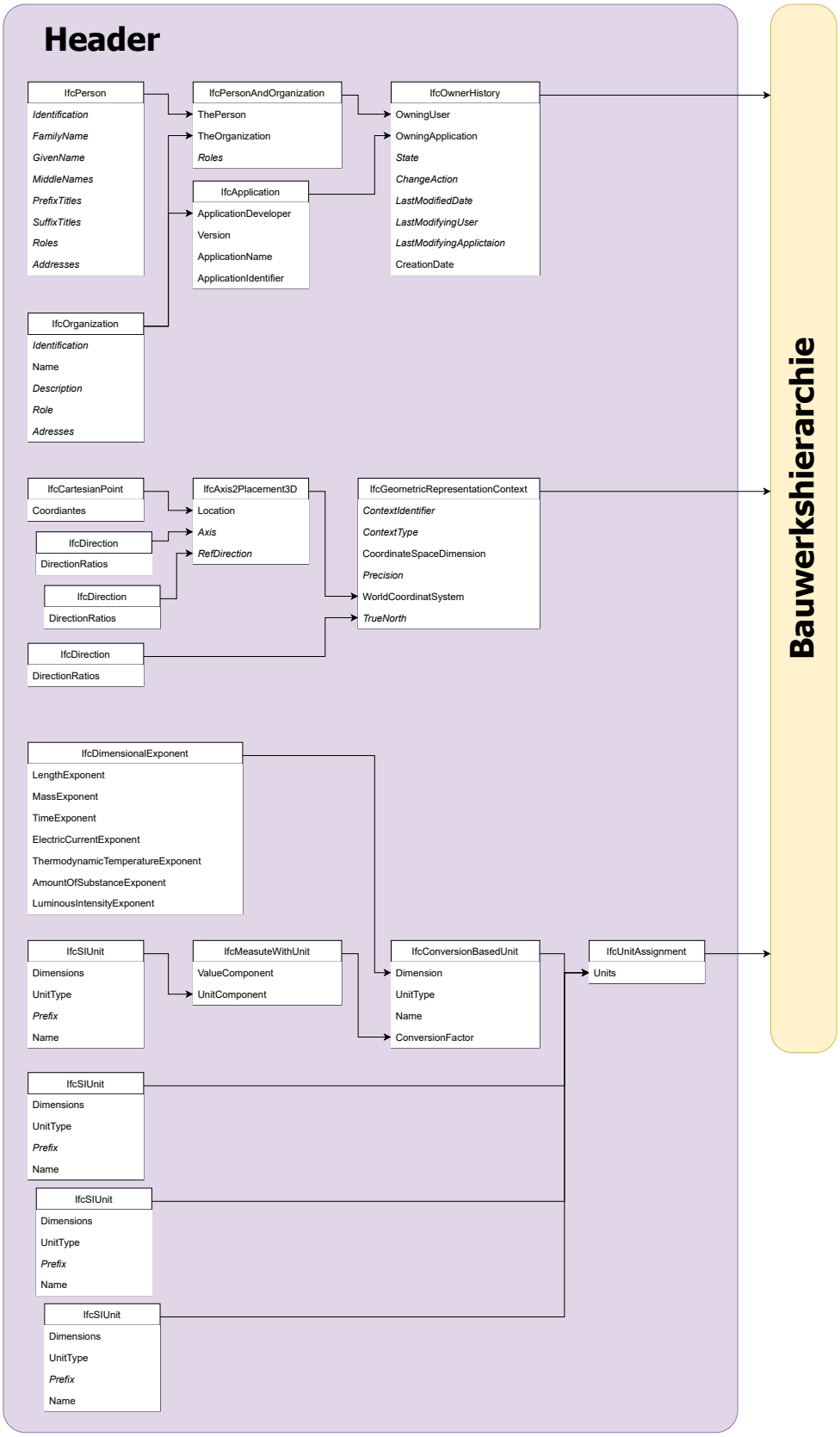


Abb. 3.5: Strukturdiagramm des Bereiches Header

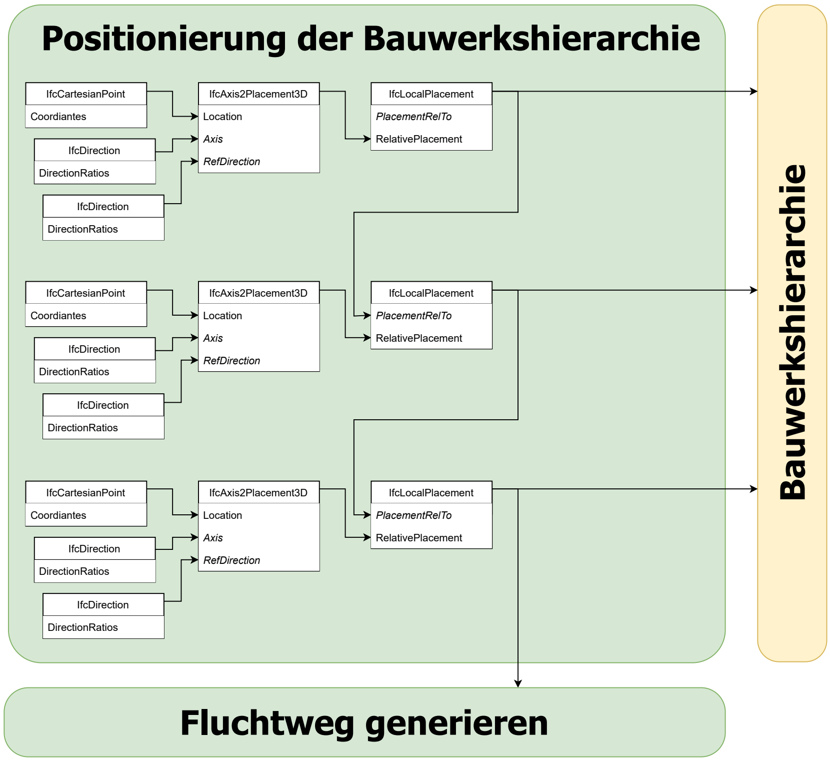


Abb. 3.6: Strukturdiagramm des Bereiches Positionierung der Bauwerkshierarchie

3.2.3 Bauwerkshierarchie und Verortung

Die Bauwerkshierarchie stellt die verschachtelten Strukturebenen eines Gebäudes dar. Sie geht vom übergeordneten Projekt bis zum einzelnen Geschoss. Alle anderen Bereiche der Ausgangsstruktur dienen dazu, diese Hierarchieebenen richtig zu verknüpfen. Das *IfcProject* stellt den Ausgangspunkt für die gesamte Datei dar. Diese Entity wird mit allen Informationen des Header-Bereiches befüllt und wendet diese auf die restlichen Entities an. Unter *IfcProject* werden die Ebenen *IfcSite*, *IfcBuilding* und *IfcBuildingStorey* angelegt. Jede dieser Entities wird über den Bereich Positionierung der Bauwerkshierarchie im Raum platziert. Die einzelnen Positionen hängen hierarchisch voneinander ab. Der Bereich Verortung ergänzt schlussendlich die Verbindung zwischen den einzelnen Entities. Er beschreibt die Zusammenhänge der einzelnen Elemente der Bauwerkshierarchie. In diesem Bereich wird somit die Reihenfolge der Hierarchieebenen festgelegt. Die Entity *IfcRelAggregates* verbindet jeweils zwei Ebenen miteinander. Von dieser sind für die gesamte Bauwerksstruktur genau drei Instanzen erforderlich. Abbildung 3.7 stellt die Bereiche Bauwerkshierarchie und Verortung vollständig dar.

3.3 Umsetzung der Konzepte in der IFC-Datenstruktur

Die in Abschnitt 3.1 beschriebenen Konzepte zur Modellierung von Fluchtwegen werden nun in fünf verschiedenen Varianten umgesetzt. Für jede dieser Ausführungen beschreibe ich die einzelnen Strukturdiagramme, welche die Umsetzung im IFC-Schema aufzeigen. Hierbei sind in den Diagrammen immer die Änderungen zur vorhergehenden Variante hervorgehoben.

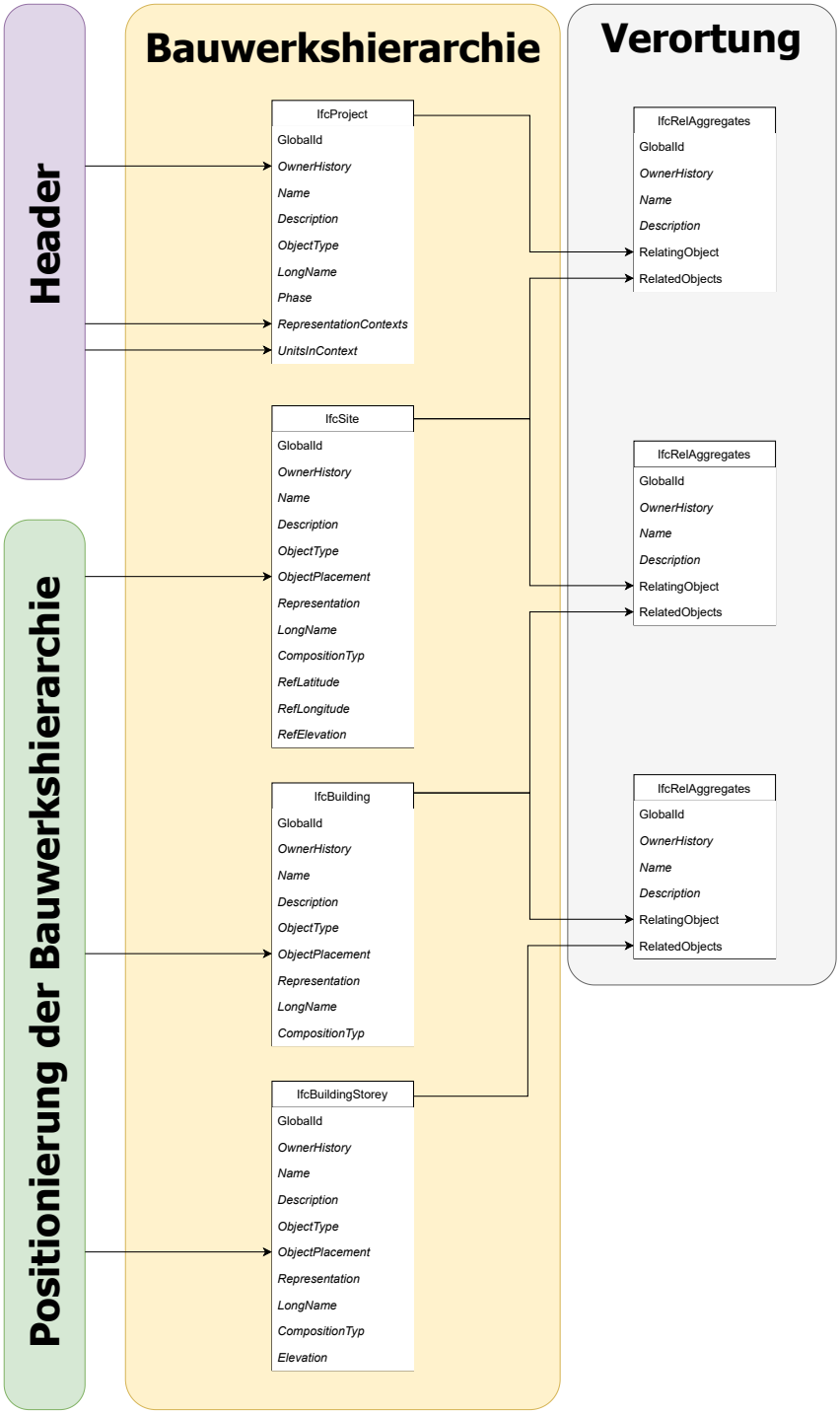


Abb. 3.7: Strukturdiagramm der Bereiche Bauwerkshierarchie und Verortung

3.3.1 Gesamte Fluchtwege als ein Element – Variante Annotation

Die erste Modellierungsvariante stellt einen ganzen Fluchtweg als ein Element dar. Abbildung 3.8 zeigt den Aufbau der IFC-Datei mit allen zusätzlichen Entities. Das Grundgerüst der IFC-Datei aus Abschnitt 3.2 bleibt ohne Änderungen vorhanden. Das Modell verwendet die Entity *IfcAnnotation* [6], um die einzelnen Fluchtwege darzustellen. Für jedes dieser Objekte ist die Definition einer geometrischen Form und eine Positionierung notwendig. Die Positionierung wird durch ein *IfcLocalPlacement* ermöglicht. Für die geometrische Darstellung bilden mehrere 3D-Koordinatenpunkte der Entity *IfcCartesianPoint* zusammen eine *IfcPolyline*. Da in diesem Modell eine 3D-Annotation den Fluchtweg darstellt, ist kein Volumenkörper als Geometrie notwendig. Eine 3D-Polylinie genügt in diesem Fall. In den Entities *IfcPropertySingleValue* speichert das Modell die Fluchtwegslänge, Personenanzahl und Fluchtwegsnamen ab. Diese einzelnen Merkmale fasst die Entitiy *IfcPropertySet* zusammen. Schlussendlich verknüpft das Element *IfcRelDefinesByProperties* dieses PSet zu der eigentlichen Annotation. Damit sind die Informationen eindeutig einem Fluchtwegselement zugeordnet. Um den Fluchtweg in der Bauwerksstruktur zu verorten, wird die Entity *IfcRelContainedInSpatialStructure* verwendet. Mit dieser kann ein Objekt einer Ebene der Bauwerkshierarchie zugewiesen werden.

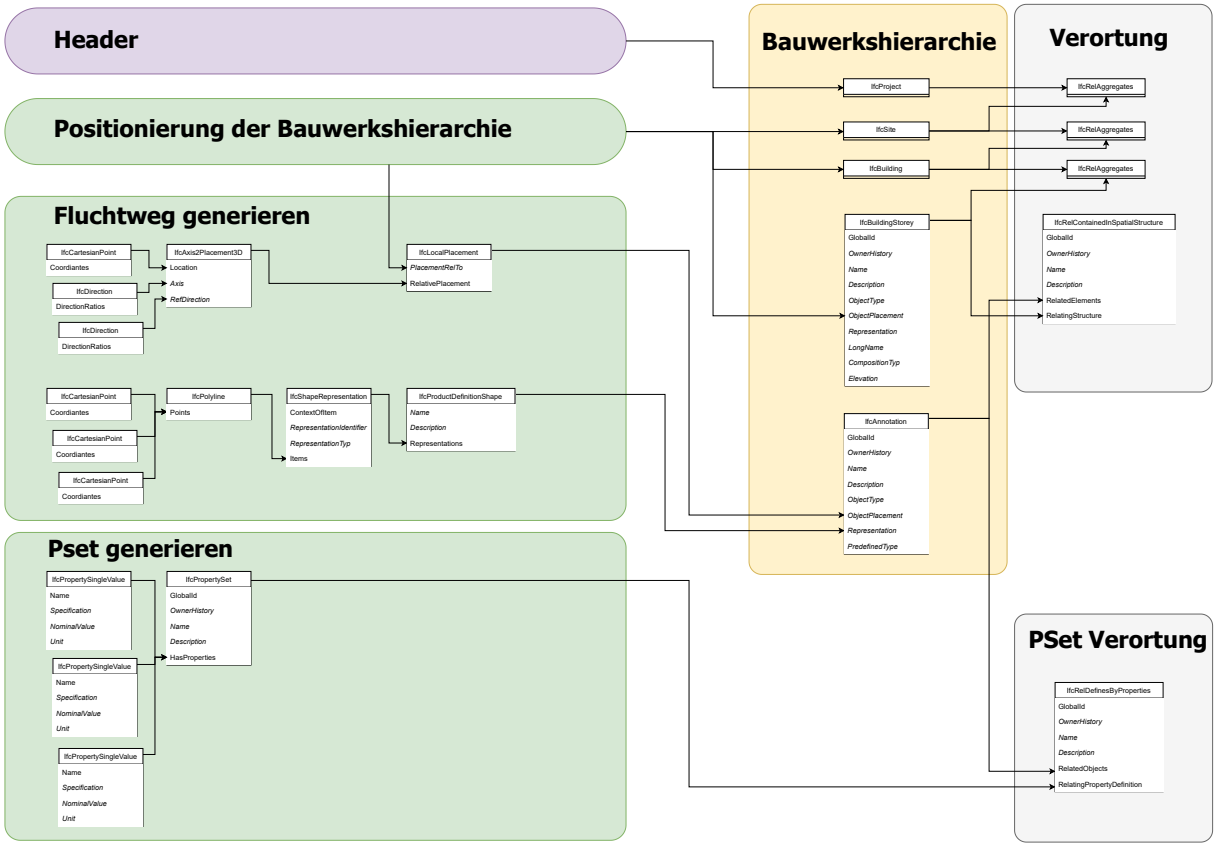


Abb. 3.8: Strukturdiagramm der Modellierungsvariante „Annotation“. Das Grundgerüst der IFC-Datei aus Abschnitt 3.2 bleibt vorhanden und ist hier nur abstrakt dargestellt.

3.3.2 Fluchtwege als einzelne Segmente – Variante Proxy

Als Alternative zur Variante Annotation kann jeder Fluchtweg in einzelne Segmente zerlegt werden. Diese stellen gemeinsam wieder den gesamten Pfad dar. Um diese Modellierung zu realisieren, wird

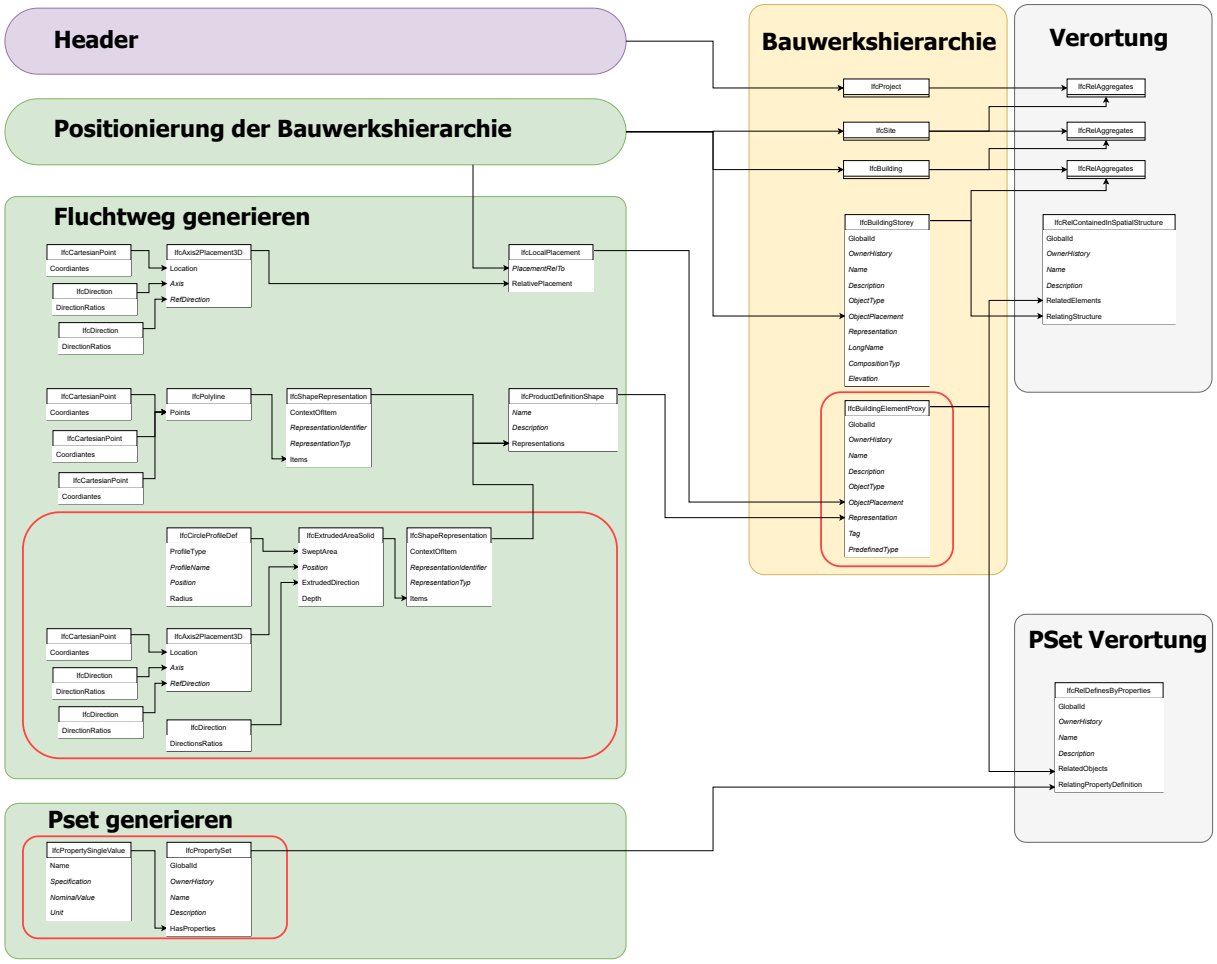


Abb. 3.9: Strukturdiagramm der Modellierungsvariante „Proxy“

die Entitiy *IfcElementProxy*[1] für die Fluchtwegssegmente verwendet. Im Vergleich zur Variante Annotation zeigt Abbildung 3.9 die zusätzlich nötigen Entities in der IFC-Datei. Die *IfcPolyline* stellt nun nicht mehr den gesamten Fluchtwegspfad, sondern nur die Achse eines einzelnen Segmentes dar. Für die Proxy-Elemente wird neben der Polylinie ein Volumenkörper als Geometrie benötigt. Dafür dient ein Zylinder mit einem vordefinierten Radius. Um diesen zu generieren, wird ein Flächenprofil als *IfcCircleProfileDef* erstellt. Die Entity *IfcExtrudedAreaSolid* extrudiert daraus einen Volumenkörper. Die Achse des Segmentes und der Zylinder bilden gemeinsam die *IfcProductDefinitionShape*, welche das *IfcElementProxy* definieren. Die Personenanzahl je Segment wird als *IfcPropertySingleValue* an das Proxyelement geknüpft.

3.3.3 Fluchtwege als einzelne Segmente – Variante Property

Befinden sich mehrere unabhängige Fluchtwege in derselben IFC-Datei, muss die Variante Proxy weiter verfeinert werden. Die Segmente benötigen ein zusätzliches Merkmal, um sie den richtigen Fluchtwegen zuordnen zu können. Hierfür wird eine zusätzliche Entity vom Typ *IfcPropertySingelValue* an die Proxyelemente gehängt. Dabei kann ein einzelnes Segment mehreren Fluchtwegen zugeordnet sein. Hierfür wird die Entity *IfcPropertyListValue* verwendet. Die

[1] Es ist für alle Varianten möglich, die Entity *IfcAnnotation* statt *IfcElementProxy* für die Fluchtwegssegmente zu verwenden. Hierbei würde die Modellierung eines Volumenkörpers für jedes Segment entfallen.

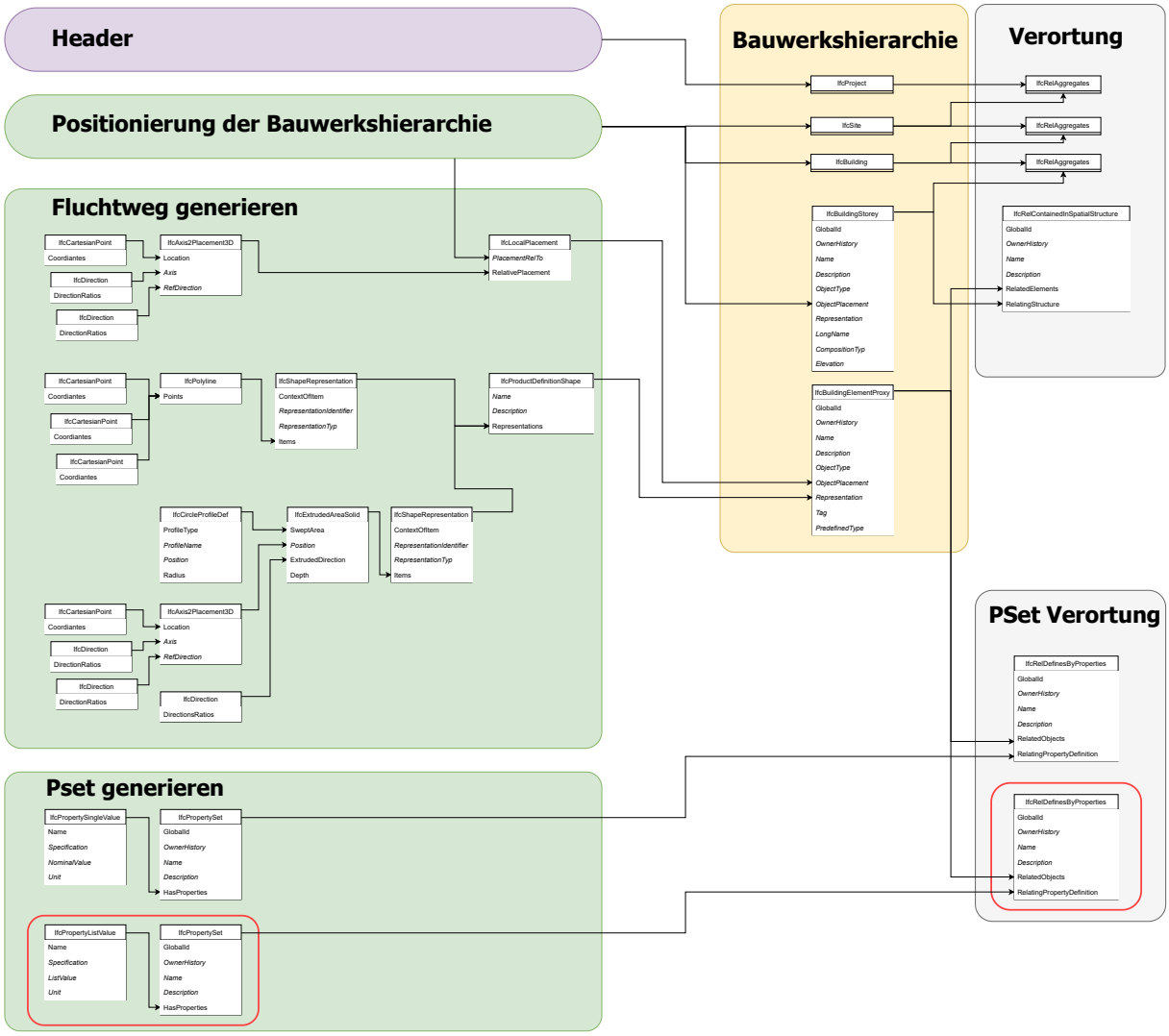


Abb. 3.10: Strukturdiagramm der Modellierungsvariante „Property“

Verknüpfung zu dem Proxy-Element erfolgt wieder über den Befehl *IfcRelDefinesByProperties*. Abbildung 3.10 zeigt die Erstellung dieses zusätzlichen PSets.

3.3.4 Fluchtwege als Gruppierung mehrerer Segmente – Variante Group

Die Zuordnung der einzelnen Segmente zu bestimmten Fluchtwegen kann auch ohne ein zusätzliches Merkmal erreicht werden. Abbildung 3.11 zeigt die Verwendung einer *IfcGroup*. Diese Entity ermöglicht die übergeordnete Verknüpfung mehrerer Objekte zu einem neuen Group-Element. Mit dem Befehl *IfcRelAssignsToGroup* werden die einzelnen Proxy-Elemente in einer Gruppe eingebettet. Ein Proxy-Element kann dabei mehreren Gruppen zugeordnet sein. Die Verortung der *IfcGroup* in der Bauwerkshierarchie stellt einen Spezialfall dar. Die Verwendung der Entity *IfcRelContainedInSpatialStructure*, wie bei den anderen Varianten, ist hier nicht möglich. Es muss die Entity *IfcReferencedInSpatialStructure* eingesetzt werden. Normalerweise ermöglicht diese Entity die zusätzliche Verortung eines Objekts abseits seiner eigentlichen Position in der Bauwerksstruktur. Zum Beispiel kann ein Liftschacht damit mehreren Geschossen gleichzeitig zugeordnet sein. Nur der Sonderfall einer *IfcGroup* verlangt diese Entity als einzige Verortung.

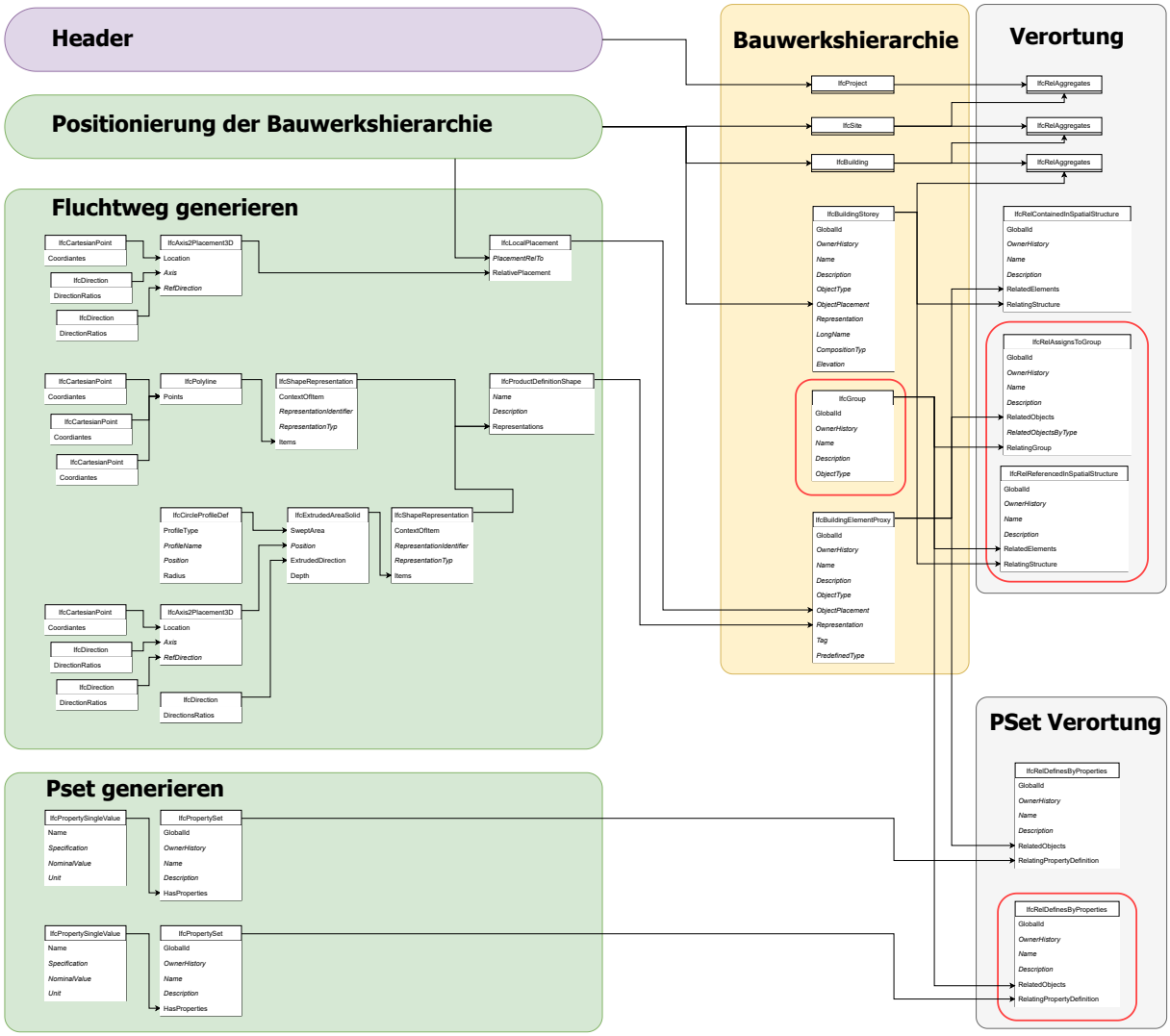


Abb. 3.11: Strukturdiagramm der Modellierungsvariante „Group“

Schlussendlich können durch diese Maßnahme die einzelnen Fluchtwegssegmente in einer anderen Hierarchieebene liegen als die Gruppe, welche den gesamten Fluchtweg beschreibt. Als Merkmale werden die Personenanzahl den Segmenten, und die Fluchtwegslänge den Gruppen zugeordnet. Hierfür werden zwei Entities vom Typ *IfcPropertySingleValue* verwendet.

3.3.5 Fluchtwege als Gruppierung mehrerer Segmente – Variante Part

Die Modellierung des Konzeptes „Fluchtwege als Gruppierung mehrerer Segmente“ kann auch ohne die Entity *IfcGroup* gelöst werden. Abbildung 3.12 zeigt die Verwendung der Entity *IfcBuildingElementPart*. Diese stellt nun die einzelnen Segmente anstelle der Proxy-Elemente dar. Über den Befehl *IfcRelAggregates* können mehrere Parts einem anderen Bauteil zugeordnet werden. Die Entity *IfcBuildingElementProxy* dient in diesem Fall als gesamter Fluchtweg, welcher aus den einzelnen Segmenten als Part-Elementen besteht. Das Proxy- und die Part-Elemente werden jeweils über die Entity *IfcRelContainedInSpatialStructure* in der Bauwerksstruktur verortet. Die Merkmale werden wieder als PSets gespeichert. Die Fluchtwegslänge wird den Proxy-Elementen, und damit den gesamten Fluchtwegen, und die Personenanzahl den einzelnen Parts zugeordnet.

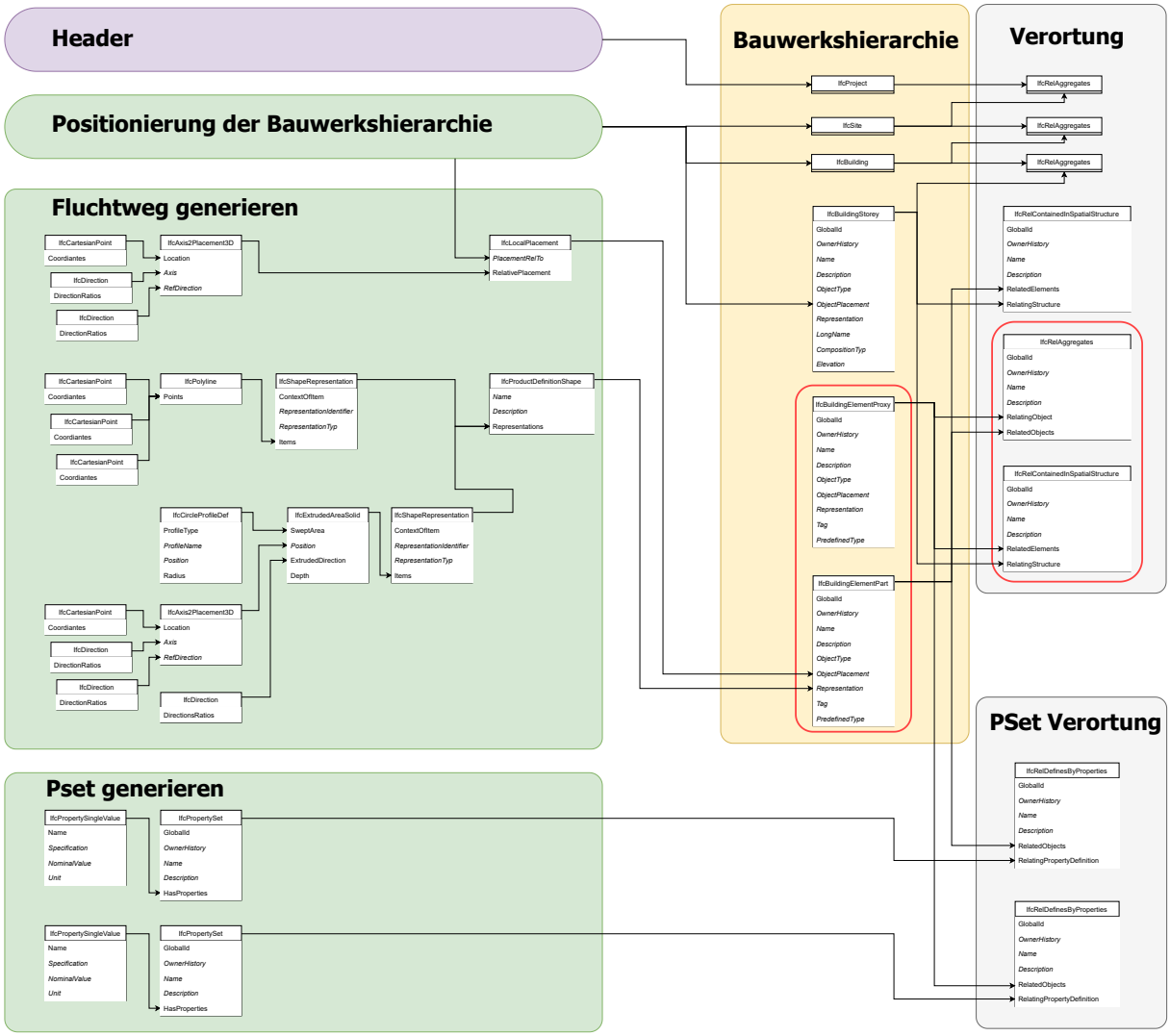


Abb. 3.12: Strukturdiagramm der Modellierungsvariante „Part“

Kapitel 4

IfcOpenShell

Dieses Kapitel behandelt die zweite der beiden Forschungsfragen: wie kann die Generierung der Fluchtwegsmodelle effizient automatisiert werden? Als Leitfaden verwende ich hierbei meinen Programmcode, welcher aus einer Eingabedatei ein vollständiges Fluchtwegsmodell generiert. Im ersten Schritt ermöglicht Python, die Eingangsdaten einzulesen und für die Bearbeitung vorzubereiten. Aus diesen Daten erstellt das Programm eine erste Vorlagedatei im IFC-Format. Die IfcOpenShell-Bibliothek kann danach neue Objekte zu dieser Datei hinzufügen. Zuerst wird die Bauwerkshierarchie und anschließend die einzelnen Fluchtwege generiert. Danach positioniert das Programm die Fluchtwege richtig in der Bauwerksstruktur. Schlussendlich werden Informationen wie Fluchtwegslängen und Personenzahlen als PSets ergänzt.

4.1 Anwendung von Python

Das Programm verwendet abseits der IfcOpenShell-Bibliothek auch viele Funktionen der Python-Programmiersprache. In diesem Abschnitt beschäftige ich mich mit der Verarbeitung der Eingangsdaten und der Erstellung einer Vorlagedatei im IFC-Format, welche als Grundlage für die weitere Bearbeitung dient.

4.1.1 Verarbeitung der Eingangsdaten

Aus der vorangestellten Fluchtwegsberechnung liegt als Ausgangspunkt eine JSON-Datei vor. In dieser Datei gibt es vier primäre Datensätze: Bauplatz (site), Gebäude (building), Geschosse (storey) und Fluchtwege (path). Die Bauplatzdaten enthalten die absolute Höhe, das Projektnull, den Projektnamen und den geografischen Längen- und Breitengrad. In den Gebäudedaten ist für jedes Gebäude der lokale Nullpunkt, bezogen auf das Projektnull, sowie der lokale Gebäudenullpunkt angegeben. Im dritten Abschnitt der Datei werden die Geschosshierarchien aufgelistet. Für jedes Geschoss ist der Name, das zugehörige Gebäude und der lokale Nullpunkt des Geschosses bezogen auf das Gebäudenull angegeben. Zusätzlich ist die Höhe des Geschosses noch einmal explizit enthalten. Die Informationen der Fluchtwege gliedern sich in Objektdaten und Geometriedaten. Erstere beinhalten Namen, Länge sowie die Anzahl der flüchtenden Personen. Außerdem ist jenes Stockwerk hinterlegt, in dem der Fluchtweg seinen Ursprung hat. Die Geometriedaten bestehen aus einer Abfolge von 3D-Koordinaten, welche den Verlauf des Fluchtweges darstellen. Das Übergabefile beinhaltet diese Koordinatenpunkte bereits in der geometrisch richtigen Reihenfolge.

Das Programm liest die Modell- und Fluchtwegsdaten mit Hilfe des integrierten JSON-Package [29] ein. Anschließend erfolgt wie in Abbildung 4.1 eine Aufteilung der Daten in die Kategorien *site*, *building*, *storey* und *geometry*. Die Daten der *site* Kategorie kann ich direkt herausfiltern, da immer von genau einem Projekt je Eingabedatei ausgegangen wird. Ein einfacher Algorithmus speichert die restlichen Gebäudeinformationen in den Gruppen *building* und *storey*. Dieser Prozess ist in Programmcode 4.1 dargestellt.

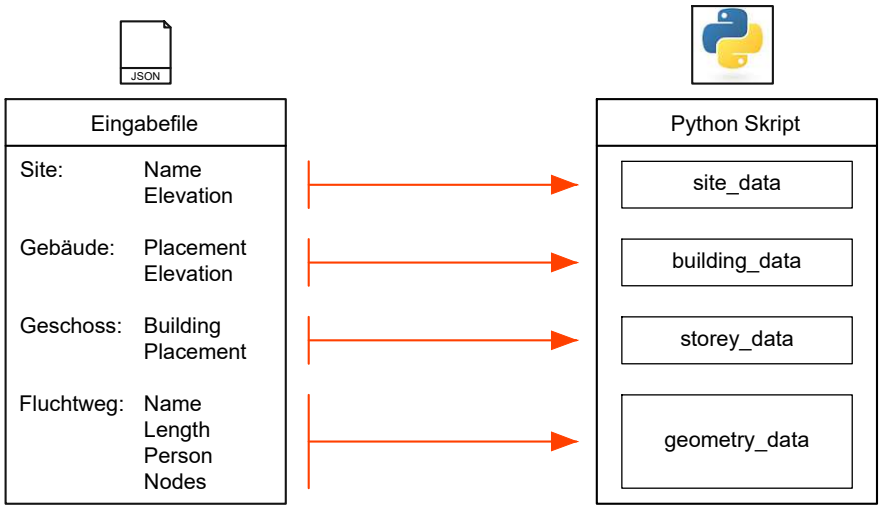


Abb. 4.1: Verarbeitung der Eingangsdaten aus der JSON-Datei

```
# Vorbereiten der Daten
try:
    data = json.load(file_json)
    site_data = data[0]
    building_data = data[1]
    storey_data = data[2]
    geometry_data = data[3:]
except IndexError:...

# site data auslesen und neu abspeichern als Site Objekt
site_latitude = (site_data['Latitude']['Degrees'],
                 site_data['Latitude']['Seconds'],
                 site_data['Latitude']['Minutes'],
                 site_data['Latitude']['MicroSeconds'])
site_longitude = (site_data['Longitude']['Degrees'],
                  site_data['Longitude']['Seconds'],
                  site_data['Longitude']['Minutes'],
                  site_data['Longitude']['MicroSeconds'])
site_obj = Site(site_data['Sitename'], site_data['Elevation'], site_latitude,
                site_longitude)

# building data auslesen und neu abspeichern
buildings_obj = []
for key in list(building_data.keys()):
    # für jedes Gebäude wird ein Building Objekt erstellt
    building_obj = Building(key,
                            building_data[key]['Elevation'],
                            building_data[key]['relativePlacement'])
    buildings_obj.append(building_obj)

# storey data auslesen und neu abspeichern
storeys_obj = []
for key in list(storey_data.keys()):
    # für jedes Stockwerk wird ein Storey Objekt erstellt
    storey_obj = Storey(key,
                        storey_data[key]['Building'],
                        storey_data[key]['Elevation'],
                        storey_data[key]['relativePlacement'])
    storeys_obj.append(storey_obj)
```

Programmcode 4.1: Verarbeitung der Eingangsdaten für die Bauwerkshierarchie

Das Programm übernimmt die Objektdaten der Fluchtwege direkt aus dem Übergabefile. Es erfolgt abseits einer lokalen Zuordnung der Daten keine Veränderung. Je nach weiterer Verwendung muss ein Algorithmus wie in Abbildung 4.2 die Geometriedaten bearbeiten, damit keine zwei identischen Fluchtwegsabschnitte entstehen. Das Programm überprüft jeweils die Anfangs- und Endkoordinaten eines Abschnittes mit den vorherigen. Wird ein geometrisch gleicher Abschnitt entdeckt, ändert der Algorithmus lediglich die Personenanzahl, welche auf diesem Fluchtwegssegment zu berücksichtigen sind, ohne einen neuen Abschnitt zu erstellen. Aus dieser Vorbereitung folgt eine neue Datenstruktur, welche das Programm für die Erstellung einer IFC-Datei bereitstellt. Die Umsetzung dieses Algorithmus zeigt der Programmcode 4.2

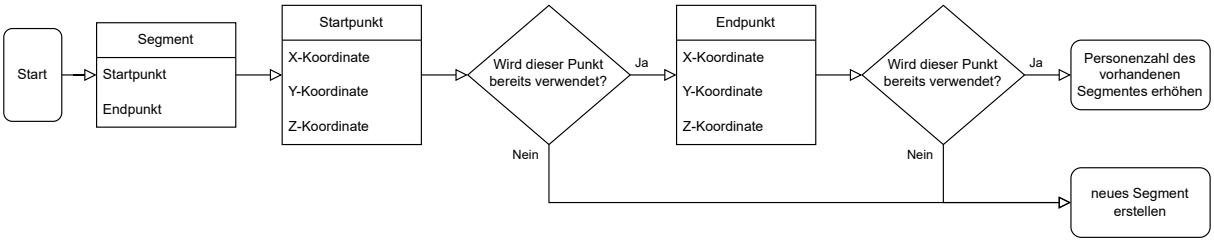


Abb. 4.2: Algorithmus zur Überprüfung doppelter Fluchtwegssegmente

```
# geometry data auslesen und neu abspeichern
fluchtwege_obj = [] # Liste aller Fluchtwege
segments_dict = dict() # Dictionary aller Segmente
for line in geometry_data:
    # Daten des Fluchtweges auslesen
    name = line['Path']
    start_storey = line['StartBuildingStorey']
    length = line['Length']
    people = line['Persons']
    nodes = line['Nodes']

    # Fluchtweg Objekt erstellen
    fluchtweg_obj = Fluchtweg(name, length, start_storey)

    # Über alle Segmente des Fluchtweges
    for i in range(len(nodes) - 1):
        # Anfangs- und Endknoten des Segmentes auslesen
        start_node = tuple(nodes[i])
        end_node = tuple(nodes[i + 1])

        # Auf idente Segmente prüfen
        if start_node in segments_dict.keys() and
           segments_dict[start_node].end == end_node:
            # Segment existiert -> Personenanzahl erhöhen
            segment = segments_dict[start_node]
            segment.people += int(people)
        else:
            # Segment neu -> neues Objekt anlegen
            segment = Segment(start_node, end_node, people)
            segments_dict[start_node] = segment

        # Segment in die lokale Liste des Objektes hinterlegen
        fluchtweg_obj.segments.append(start_node)

    # Das Objekt in die Liste aller Fluchtwege aufnehmen
    fluchtwege_obj.append(fluchtweg_obj)
```

Programmcode 4.2: Verarbeitung der Eingangsdaten der einzelnen Fluchtwege. Es erfolgt eine segmentweise Überprüfung.

4.1.2 Erstellung einer IFC-Vorlagedatei

Der Beginn einer IFC-Datei wird als Header bezeichnet. In diesem Anfangssegment sind Informationen wie Autor, Dateiname und verwendetes IFC-Schema angeführt. Nach diesem Abschnitt folgt der eigentliche Inhalt einer IFC-Datei. Das Programm befüllt die ersten 20 Zeilen mit sehr grundlegenden Informationen, wie beispielsweise *IfcOwnerHistory* und allen *IfcSIUnit*-Einträgen. Der globale Projektnullpunkt in Form eines *IfcCartesianPoint* und die globalen Achsenrichtungen sind ebenfalls in diesem Bereich angeführt. Von besonderer Bedeutung ist die Deklaration des *IfcProject*. Es symbolisiert den Ausgangspunkt der Bauwerkshierarchie und dient als Anker für die Verortung aller Objekte. Außerdem werden hiermit die zuvor getroffenen Einstellungen auf das Projekt angewendet, wie Einheiten und globaler Nullpunkt. Abbildung 4.3 zeigt den gesamten Beginn der generierten IFC-Datei. Das Programm definiert die Basisdaten für das Befüllen des Header als globale Parameter. Diese Informationen werden zu einem vorgefertigten String verbunden, welcher den gesamten Beginn der IFC-Datei darstellt. Die Python-tempfile-Bibliothek [28] übernimmt diese Angaben in eine leere Datei. Diese stellt den Ausgangspunkt für die Bearbeitung mit der IfcOpenShell-Bibliothek dar.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('ViewDefinition [CoordinationView]'),'2;1');
FILE_NAME('Fluchtweg.ifc','2023-08-31T15:15:55',('Matthias Haselberger'),('TU Wien'),'IfcOpenShell','IfcOpenShell','');
FILE_SCHEMA(('IFC4'));
ENDSEC;
DATA;
#1=IFCPERSON($,$,'Matthias Haselberger',$,$,$,$,$);
#2=IFCORGANIZATION($,'TU Wien',$,$,$);
#3=IFCPERSONANDORGANIZATION(#1,#2,$);
#4=IFCAPPLICATION(#2,'0.7','IfcOpenShell','');
#5=IFCOWNERHISTORY(#3,#4,$,.NOCHANGE.,$,#3,#4,1693494955);
#6=IFCDIRECTION((1.,0.,0.));
#7=IFCDIRECTION((0.,0.,1.));
#8=IFCCARTESIANPOINT((0.,0.,0.));
#9=IFCAXIS2PLACEMENT3D(#8,#7,#6);
#10=IFCDIRECTION((0.,1.));
#11=IFCGEOMETRICREPRESENTATIONCONTEXT($,'Model',3,1.E-05,#9,#10);
#12=IFCDIMENSIONALEXPONENTS(0,0,0,0,0,0,0);
#13=IFCSIUNIT(*,.LENGTHUNIT.,$,.METRE.);
#14=IFCSIUNIT(*,.AREAUNIT.,$,.SQUARE_METRE.);
#15=IFCSIUNIT(*,.VOLUMEUNIT.,$,.CUBIC_METRE.);
#16=IFCSIUNIT(*,.PLANEANGLEUNIT.,$,.RADIAN.);
#17=IFCMEASUREWITHUNIT(IFCPLANEANGLEMEASURE(0.017453292519943295),#16);
#18=IFCCONVERSIONBASEDUNIT(#12,.PLANEANGLEUNIT.,'DEGREE',#17);
#19=IFCUNITASSIGNMENT((#13,#14,#15,#18));
#20=IFCPROJECT('1aeGmXXUDB5hVuUgxeoDlp',#5,'EscapeRouteAnalysis',$,$,$,$,(#11),#19);
```

Header

Fest einprogrammierter Beginn der IFC-Datei

Abb. 4.3: Darstellung des Beginns einer IFC-Datei mit dem Header und den vom Programm definierten ersten 20 Einträgen

4.2 Anwendungen der IfcOpenShell-Bibliothek

Für den weiteren Ablauf verwendet das Programm von nun an die IfcOpenShell-Bibliothek. Diese kann eine Fülle an neuen Einträgen in einer IFC-Datei generieren. Der Befehl `ifcopenshell.open` öffnet die zuvor erstellte Vorlagedatei. Diese Datei wird unter der Variable `ifcfile` abgelegt. Nach vollendeter Bearbeitung wird der Inhalt über den `ifcopenshell.write` Befehl als eine neue IFC-Datei abgespeichert.

4.2.1 Generierung der Bauwerkshierarchie

Die Bauwerkshierarchie beschreibt den Zusammenhang zwischen Grundstück, Gebäude, Geschoss und allen räumlichen Elementen in der IFC-Datei. Jedes Bauteil wird als eine eigene IFC-Entity implementiert. Ich muss zusätzlich für jedes dieser Objekte eine räumliche Orientierung und einen Zusammenhang mit den anderen Elementen definieren. *IfcLocalPlacement* beschreibt die Positionierung des Objekts anhand eines Koordinatenpunktes und einer Ausrichtung. Diese Platzierung kann relativ zu einer anderen Entity oder direkt auf eine globale Koordinate erfolgen. *IfcRelAggregates* regelt den hierarchischen Zusammenhang der einzelnen Entities. Mit diesem Befehl können einzelne Elemente verschachtelt werden. *IfcSite* wird als Ausgangspunkt für die Bauwerkshierarchie gewählt. Diese Entity gibt es per Definition in dem Projekt genau ein einziges Mal. Das *IfcLocalPlacement* verwendet den globalen Nullpunkt der IFC-Datei. Der Befehl *IfcRelAggregates* wird mit dem Projekteintrag im Header verschachtelt. Die nachfolgenden Hierarchieebenen *Buildings* und *Storeys* erstellt das Programm nach dem selben Schema. Dabei verschachtelt es die Entity *Building* zum *Site* und die Entity *Storey* zum *Building*. Daraus entsteht die finale Projekthierarchie (siehe Abbildung 4.4), in welcher die generierten Fluchtwege platziert werden können. Programmcode 4.3 zeigt diesen Prozess beispielhaft für die Hierarchieebene eines Gebäudes.

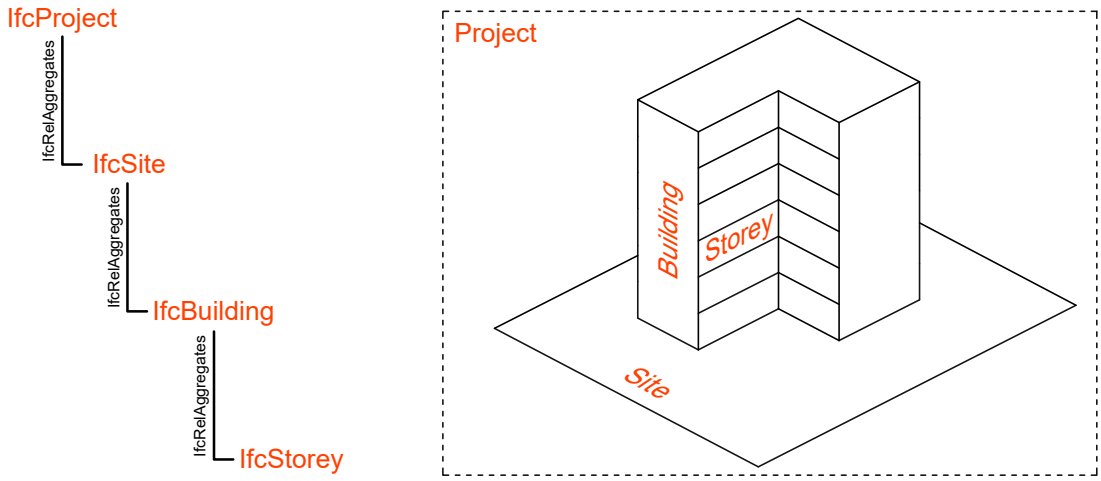


Abb. 4.4: Erstellung der Bauwerkshierarchie für das neue IFC-Modell

```
# — Building —
# Für alle Gebäude
for building_obj in buildings_obj:
    # Erstellt ein LocalPlacement
    point = ifcfile.createIfcCartesianPoint(building_obj.placement)
    dir1 = ifcfile.createIfcDirection((0., 0., 1.))
    dir2 = ifcfile.createIfcDirection((1., 0., 0.))
    axis2placement = ifcfile.createIfcAxis2Placement3D(point, dir1, dir2)
    building_placement = ifcfile.createIfcLocalPlacement(site_placement,
                                                         axis2placement)

    # Erstellt die Entity IfcBuilding
    building = ifcfile.createIfcBuilding(ifcopenshell.guid.new(),
                                         owner_history,
                                         building_obj.name,
                                         None,
                                         None,
                                         building_placement,
                                         None,
```

```
                                        None,
                                        'ELEMENT')

    # Verknüpfung zwischen Building und Site über IFCRelAggregates
    ifcfile.createIfcRelAggregates(ifcopenshell.guid.new(),
                                   owner_history,
                                   'Building Container',
                                   None,
                                   site,
                                   [building])
```

Programmcode 4.3: Erstellung der Bauwerkshierarchie. Hier wird nur die Erstellung einer Gebäudeebene gezeigt. Die Generierung der restlichen Hierarchieebenen erfolgt nach dem selben Prinzip

4.2.2 Erstellen der Fluchtwege

Es kommen mehrere Möglichkeiten zur Erstellung der Fluchtwege in Frage (Kapitel 3). Die grundlegende Idee des Programmablaufes bleibt jedoch bei all diesen Varianten identisch. Nachfolgend beziehe ich mich auf die Modellierungsvariante „Group“. Ich erstelle im ersten Schritt aus den Geometriedaten der Fluchtwege neue 3D-Objekte in der IFC-Datei. Jedes 3D-Element benötigt eine Definition seiner Position und seines Volumens. Die Geometriedaten liegen aus der eingelesenen JSON-Datei in den lokalen Koordinaten des Gebäudes vor. Die Position des Fluchtweges lässt sich wie in Abbildung 4.5 dargestellt durch ein *IfcLocalPlacement* bestimmen, welches vom Gebäudenullpunkt ausgeht. Durch die Bauwerkshierarchie ist das Gebäudenull auf das Projektnull verknüpft, und damit den globalen Nullpunkt. Das Programm erstellt das *IfcLocalPlacement* mit dem Befehl `ifcfile.createIfcLocalPlacement(...)` und speichert es unter der Variable `proxy_placement` ab (Programmcode 4.4).

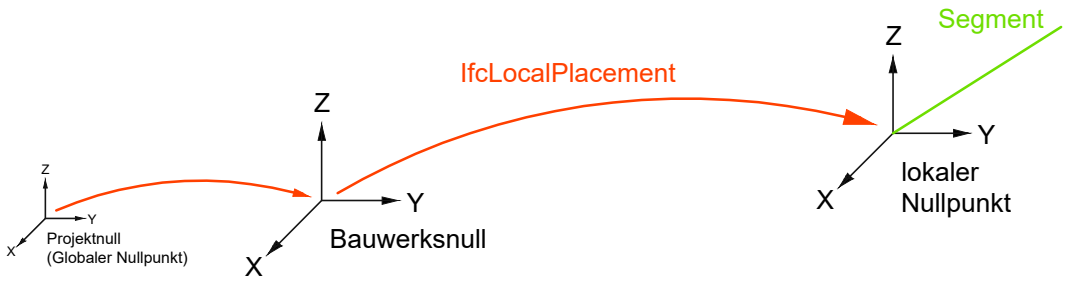


Abb. 4.5: Definition des lokalen Nullpunktes eines Fluchtwegssegmentes

```
# Anfangs- und Endpunkt des Segments
start_point = segment.start
end_point = segment.end

# Lokales Koordinatensystem für das Segment
point = ifcfile.createIfcCartesianPoint(start_point)
dir1 = ifcfile.createIfcDirection((0., 0., 1.))
dir2 = ifcfile.createIfcDirection((1., 0., 0.))
axis2placement = ifcfile.createIfcAxis2Placement3D(point, dir1, dir2)
ifclocalplacement = ifcfile.createIfcLocalPlacement(building_placement,
                                                    axis2placement)
```

Programmcode 4.4: Erstellung des lokalen Koordinatensystems für ein Fluchtwegssegment

Um ein Volumen zu generieren, verwendet das Programm die Extrusion einer 2D-Fläche entlang einer Achse. Aus dem Start- und Endpunkt eines Fluchtwegssegmentes erstellt es eine *IfcPolyline*. Diese Linie muss für die Generierung der Extrusion nicht als eigene Entity in der IFC-Datei vorhanden sein. Aus dem Gesichtspunkt der Datensicherung scheint es jedoch sinnvoll, die Information der zu Grunde gelegten 2D-Achse des Fluchtweges im Rahmen des IFC-Datenschemas abzuspeichern. Für die Extrusion muss zuerst ein lokales Koordinatensystem wie in Abbildung 4.6 erstellt werden. Als Konvention wird die Z-Achse in Extrusionsrichtung definiert und entspricht daher der Verbindungslinie zwischen Anfangs- und Endpunkt des Segmentes. Ich definiere das zu extrudierende Profil in der X-Y-Ebene des lokalen Koordinatensystems. Um diese Ebene zu finden, erfolgt eine 90°-Drehung der lokalen Z-Achse um die globale X-Achse. Eine Transformationsmatrix wird hierfür auf den Vektor der Z-Achse angewendet. Für den Sonderfall, das die Z-Achse parallel zur globalen X-Achse steht, wird die selbe Drehung um die globale Y-Achse ausgeführt. Die gedrehte Achse ist nun orthogonal auf die lokale Z-Achse und liegt damit in der lokalen X-Y-Ebene. Die genaue Ausrichtung dieser Achse ist nicht relevant, da für die Extrusion ein Kreis verwendet wird. Der Befehl `ifcfile.createIfcAxis2Placement3D(...)` erzeugt mit diesen Informationen am Startpunkt des Fluchtwegssegmentes ein lokales Koordinatensystem. Dieses wird unter der Variable `extrusion_placement` abgelegt, wie im Programmcode 4.5 gezeigt.

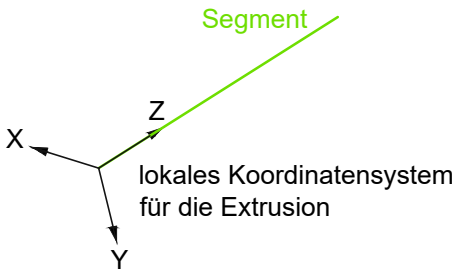


Abb. 4.6: Definition des lokalen Koordinatensystems eines Fluchtwegssegmentes

```
# Aus dem Anfangs- und Endpunkt des Segments wird eine Vektor erstellt
z_vektor = (end_point[0] - start_point[0],
            end_point[1] - start_point[1],
            end_point[2] - start_point[2])
length = math.sqrt(z_vektor[0] ** 2 + z_vektor[1] ** 2 + z_vektor[2] ** 2)
# Drehung um globale X-Achse
y_vektor = (z_vektor[0], -z_vektor[2], z_vektor[1])
# Falls parallel zur X-Achse -> Drehung um globale Y-Achse
if z_vektor == y_vektor:
    y_vektor = (-z_vektor[2], z_vektor[1], z_vektor[0])

# Achse des Segments erstellen
axis_pts = [ifcfile.createIfcCartesianPoint(start_point),
            ifcfile.createIfcCartesianPoint(end_point)]
axis = ifcfile.createIfcPolyLine(axis_pts)
axis_representation = ifcfile.createIfcShapeRepresentation(context,
                                                           'Axis',
                                                           'Curve3D',
                                                           [axis])
# Local Placement für das Volumen Element
point = ifcfile.createIfcCartesianPoint((0.0, 0.0, 0.0))
dir1 = ifcfile.createIfcDirection(z_vektor)
dir2 = ifcfile.createIfcDirection(y_vektor)
extrusion_placement = ifcfile.createIfcAxis2Placement3D(point, dir1, dir2)
```

Programmcode 4.5: Erstellung des lokalen Koordinatensystems für die Extrusion des Zylindersegmentes

Der Befehl `ifcfile.createIfcCircleProfileDef(...)` erzeugt einen Kreis in Form der Entity *IfcCircle*. Das Programm verwendet einen festgelegten Radius von 1 cm. Für die Extrusion verwendet es dieses Kreisprofil, eine Extrusionsrichtung, welche als *IfcDirection* angegeben wird, und das zuvor definierte lokale Koordinatensystem aus der Variable `extrusion_placement`. Die Extrusionsrichtung entspricht der Z-Achse wie in Abbildung 4.7. Zuletzt fehlt noch die Information der gewünschten Länge des entstandenen Körpers. Das Programm verwendet hierfür die Länge des Vektors zwischen Start- und Endpunkt des Fluchtwegssegmentes. Der Volumenkörper wird mit dem Befehl `ifcfile.createExtrudedAreaSolid(...)` erzeugt. Programmcode 4.6 zeigt das Erstellen eines Zylindersegmentes, welches schlussendlich unter der Variable `body_representation` gespeichert wird.

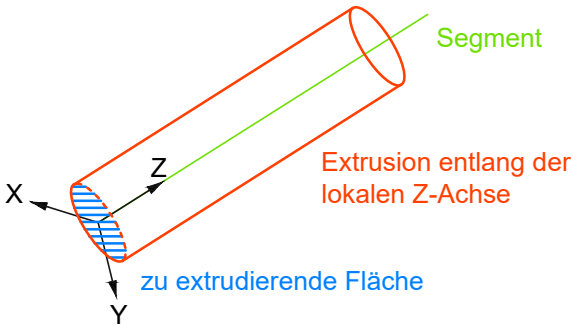


Abb. 4.7: Extrusion einer 2D-Fläche zu einem 3D-Volumenelement

```
# Extrusionsrichtung entlang der Z-Achse
extrude_dir = (0., 0., 1.)
ifccircle = ifcfile.createIfcCircleProfileDef("AREA",
                                              None,
                                              None,
                                              radius)

ifcdir = ifcfile.createIfcDirection(extrude_dir)
solid = ifcfile.createIfcExtrudedAreaSolid(ifccircle,
                                           extrusion_placement,
                                           ifcdir,
                                           extrusion)

# Erstellen des Volumenkörpers
body_representation = ifcfile.createIfcShapeRepresentation(context,
                                                           'Body',
                                                           'SweptSolid',
                                                           [solid])
```

Programmcode 4.6: Erstellen des extrudierten Volumenelementes

Mit diesem Prozess können nun aus allen Fluchtwegssegmenten 3D-Zylinderelemente generiert werden. Abbildung 4.8 zeigt, wie die einzelnen Abschnitte gemeinsam einen Fluchtweg bilden. Für jedes Segment wird ein *IfcBuildingElementProxy* generiert. Die Entity *IfcProductDefinitionShape* verknüpft Segmentachse und Zylindervolumen mit dem Proxy-Element. Die Verbindung der Segmente zu einem Fluchtweg erfolgt über die Entity *IfcRelAssignstoGroup*. Diese definiert den Inhalt der *IfcGroup*, welche den Fluchtweg darstellt. Programmcode 4.7 zeigt diesen Prozess. Abschnitt 3.3 beschreibt die unterschiedlichen Varianten dieser Zuordnung, mittels *IfcGroup* oder *IfcElementPart*. Damit sind die Geometriedaten eines Fluchtweges eindeutig in dem Datensche-

ma der IFC-Datei abgespeichert. Im nächsten Schritt füge ich den einzelnen Fluchtwegen die Objektdaten aus der JSON-Eingabedatei hinzu.

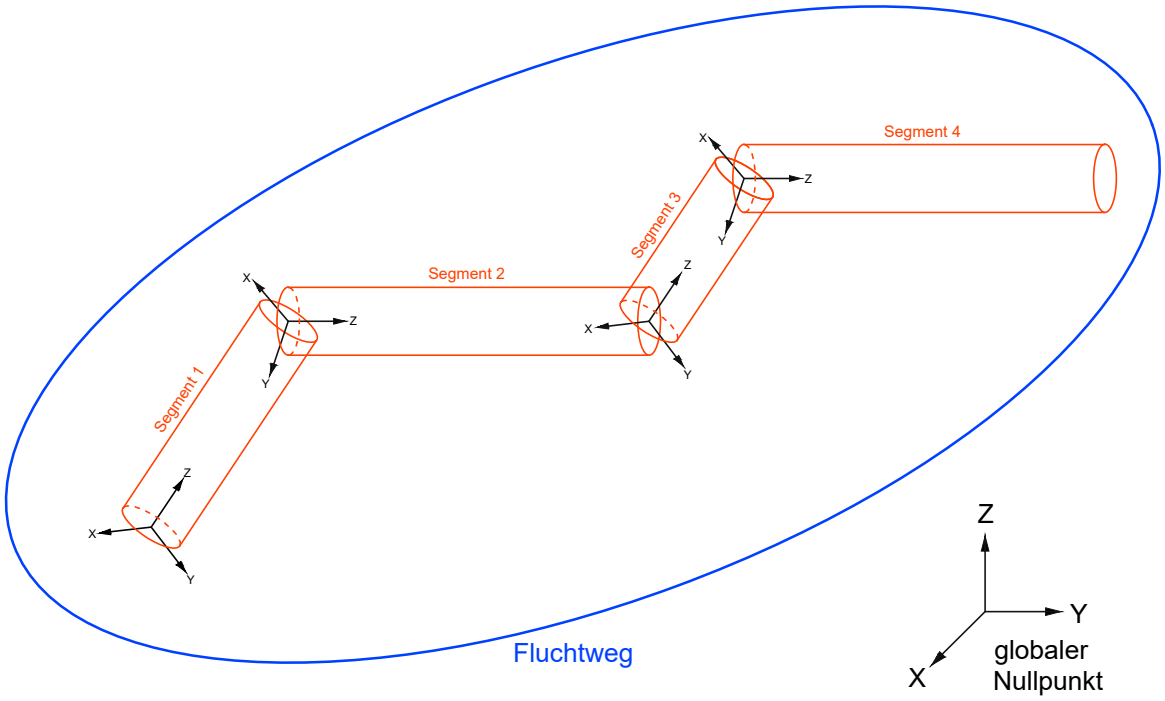


Abb. 4.8: Zusammenfassung mehrerer Segmente zu einem Fluchtweg

```
# Zusammenfassen von Volumen und Segmentachse
product_shape = ifcfile.createIfcProductDefinitionShape(None,
                                                         None,
                                                         [axis_representation,
                                                          body_representation])
# Erstellen des Proxy-Elementes
proxy = ifcfile.createIfcBuildingElementProxy(ifcopenshell.guid.new(),
                                               owner_history,
                                               'Fluchtweg-Proxy',
                                               str(segment),
                                               None,
                                               proxy_placement,
                                               product_shape,
                                               None)

# Erstellt eine IfcGroup, die den Fluchtweg bilden
group = ifcfile.createIfcGroup(ifcopenshell.guid.new(),
                               owner_history,
                               fluchtweg.name,
                               'IfcGroup',
                               None)
# Zuordnen der Segmente
ifcfile.createIfcRelAssignsToGroup(ifcopenshell.guid.new(),
                                   owner_history,
                                   'GroupAssigns for ' + fluchtweg.name,
                                   None,
                                   proxy_list,
                                   None,
                                   group)
```

Programmcode 4.7: Erstellen eines Proxy-Elementes

4.2.3 Generierung von Property Sets

Die Objektdaten jedes Fluchtweges enthalten dessen Personenanzahl, Länge und Ausgangsgeschoss. Das Programm verwendet *IfcPropertySingleValue* und *IfcPropertySet*, um diese Informationen an die generierten 3D-Elemente anzuhängen. Abbildung 4.9 zeigt, welche Merkmale den einzelnen Fluchtwegssegmenten und welche dem gesamten Fluchtweg zugeteilt werden. Ein *IfcPropertySingleValue* stellt dabei einen einzelnen Eintrag dar [8]. Dieser kann in unterschiedlichen Formen hinterlegt werden, wie zum Beispiel als Zahlenwert, Text oder logischer Wert. Er beinhaltet den Namen des Eintrages sowie dessen Information und falls erforderlich eine Einheit. Das *IfcPropertySet* fasst ein oder mehrere *IfcPropertySingleValue* zu einem Set zusammen. Der Befehl *IfcRelDefinesByProperties* verknüpft schlussendlich das *IfcPropertySet* mit einem Bauteil in der IFC-Datei. Dieser Prozess ist für ein Segment in Programmcode 4.8 abgebildet.

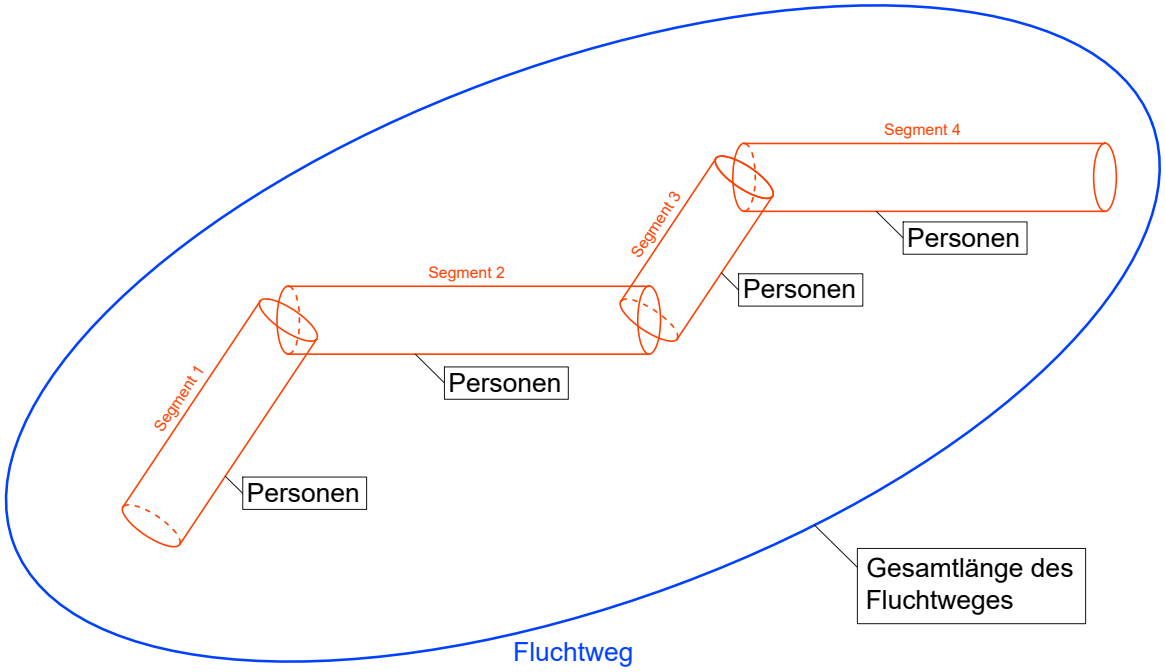


Abb. 4.9: Zuordnung von Eigenschaften über PSets

```
# Erstellen des PSet für ein Proxy-Element
peoples = ifcfile.create_entity('IfcInteger', segment.people)
property_values = [ifcfile.createIfcPropertySingleValue('People',
                                                        'People',
                                                        peoples,
                                                        None)]
property_set = ifcfile.createIfcPropertySet(ifcopenshell.guid.new(),
                                            owner_history,
                                            'Personenanzahl',
                                            'Pset_Segment',
                                            property_values)
ifcfile.createIfcRelDefinesByProperties(ifcopenshell.guid.new(),
                                        owner_history,
                                        None,
                                        None,
                                        [proxy],
                                        property_set)
```

Programmcode 4.8: Erstellung der Properties eines einzelnen Segmentes

4.2.4 Verortung der Fluchtwege in der Bauwerkshierarchie

Die erstellten 3D-Objekte sollen eindeutig in der Bauwerksstruktur verortet werden. Durch diese Zuordnung ist ein Bauteil einem bestimmten Geschoss und damit auch Gebäude zugewiesen. Das Programm behandelt hierbei die einzelnen Fluchtwegssegmente getrennt von den gesamten Fluchtwegen. Es betrachtet den Anfangsknoten eines Segments und ordnet diesen anhand der Höhe einem Stockwerk zu. Für die Verortung des gesamten Fluchtweges wird die Information über das Startgeschoss aus den Objektdaten verwendet. Abbildung 4.10 zeigt die Zuordnung des gesamten Fluchtwegs zu den einzelnen Geschossen eines Gebäudes. Das Programm verwendet den Befehl *IfcRelContainedInSpatialStructure* für die Verortung der Segmente. Für die *IfcGroup* und damit die gesamten Fluchtwege ist die Entity *IfcRelReferencedInSpatialStructure* zuständig. Programmcode 4.9 zeigt die Verortung der Objekte. Aus allen vorhandenen Fluchtwegen erzeuge ich nach diesem Schema eine 3D-Darstellung in der IFC-Datei. Im letzten Schritt verwendet das Programm den Befehl `ifcfile.write(...)`, um die IFC-Datei abzuspeichern.

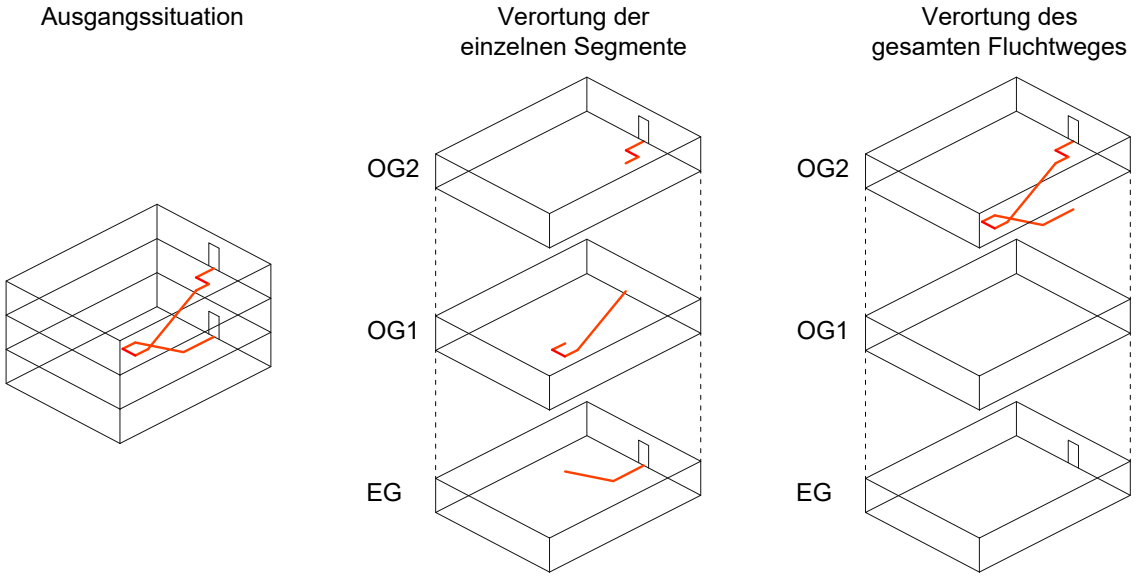


Abb. 4.10: Verortung der Fluchtwegssegmente und des gesamten Fluchtweges in der Bauwerkshierarchie

```
# Für alle Proxy- und Group-Elemente
# Verortet das Proxy-Elementes im richtigen Stockwerk
segment_elevation = segment.start[2]
ifcfile.createIfcRelContainedInSpatialStructure(ifcopenshell.guid.new(),
                                                owner_history,
                                                'Storey Container - Proxy',
                                                None,
                                                [proxy],
                                                storeys_elevation_dict[elevation])
# Verorten des Group-Elementes im richtigen Stockwerk
ifcfile.createIfcRelReferencedInSpatialStructure(ifcopenshell.guid.new(),
                                                 owner_history,
                                                 'Storey Container - Group',
                                                 None,
                                                 [group],
                                                 building_storeys_dict[
                                                   fluchtweg.start_storey])
```

Programmcode 4.9: Verortung eines Segmentes und einer Gruppe

Kapitel 5

Bewertung der unterschiedlichen Modellierungsvarianten

Die fünf verschiedenen Modellierungsvarianten (beschrieben in Abschnitt 3.3) führen gemäß des Verfahrens aus Kapitel 4 zu fünf unterschiedlichen IFC-Dateien. Zu Beginn dieses Abschnittes werden die erzeugten Modelle dargestellt. Verschiedene Aspekte dieser werden näher erläutert. Um die unterschiedlichen Varianten zu bewerten, wende ich mehrere Prüfverfahren an. Als erster Gesichtspunkt wird die grundsätzliche Qualität der entstandenen IFC-Dateien betrachtet. Hierfür wird die Syntax der Datei mit den Vorgaben des IFC-Schemas abgeglichen. buildingSMART stellt einen eigenen Validation Service [11] für IFC-Dateien als Onlinetool zur Verfügung. Das Programm IfcCheckingTool des Karlsruher Instituts für Technologie [20] verwende ich ebenfalls für diese Überprüfung. Als zweites Qualitätsmerkmal soll die Verwendbarkeit der Modelle bewertet werden. Die kommerziellen Programme Solibri [32], TrimbleConnect [35] und usBIM.viewer+ [1] zur Darstellung von IFC-Dateien kommen hierbei zum Einsatz. Danach betrachte ich die einzelnen Modelle bezüglich ihrer Erfüllung der Forschungsfragen. Schlussendlich gebe ich ein Fazit zur Verwendung der IfcOpenShell-Bibliothek. Dabei gehe ich auf die für mich entscheidenden Vor- und Nachteile ein, welche während meiner Anwendung schlagend wurden.

5.1 Generierte IFC-Modelle

Das Python Programm aus Kapitel 4 generiert eine IFC-Datei in einer der Modellierungsvarianten. Die Eingangsdaten für das Programm beziehe ich aus der Fluchtwegsberechnung nach Abschnitt 2.4. Fischer et al. [14] stellen ein Testmodell [15] für eine Fluchtwegsberechnung in ihrer Arbeit zur Verfügung. Dieses Modell ist in Abbildung 5.1 in den drei verwendeten Programmen dargestellt. Es wurde speziell für die Validierung der automatisierten Berechnung als Teil des Projekts BRISE-Vienna [34, 36] entwickelt. Das Gebäudemodell besitzt fünf Stockwerke inklusive einer Tiefgarage. Spezielles Augenmerk wurde auf die Integration von mehreren möglichen Fluchtwegen je Raum, Wegstrecken durch mehrere Räume und die Einbindung von Parkplatzflächen gelegt. Das neu generierte Fluchtwegsmodell kann später gemeinsam mit dem Testmodell dargestellt werden.

Nach der Bearbeitung jeder Modellierungsvariante liegen als Ergebnis dieser Arbeit fünf verschiedene Fluchtwegsmodelle vor. Diese erscheinen dabei optisch ähnlich, unterscheiden sich jedoch, wie in Abschnitt 3.3 beschrieben, in der genauen Aufteilung der Segmente und deren Merkmalen. Im Modell der Variante „Annotation“ existieren 18 Fluchtwege als *IfcAnnotation*, welche gemeinsam 54 unterschiedliche Merkmale in Form von *IfcPropertySingleValue* aufweisen. „Proxy“ und „Property“ enthalten keine Objekte für die Fluchtwege, jedoch jeweils 152 Segmente. Durch die Zuordnung der Segmente über Merkmale besitzt die Variante „Property“ 308 Merkmale. „Group“ und „Part“ sind in der Anzahl ihrer Objekte ident, diese unterscheiden sich jedoch in den verwendeten Entities. Eine vollständige Übersicht der schlussendlich generierten Objekte ist in Tabelle 5.1 dargestellt.

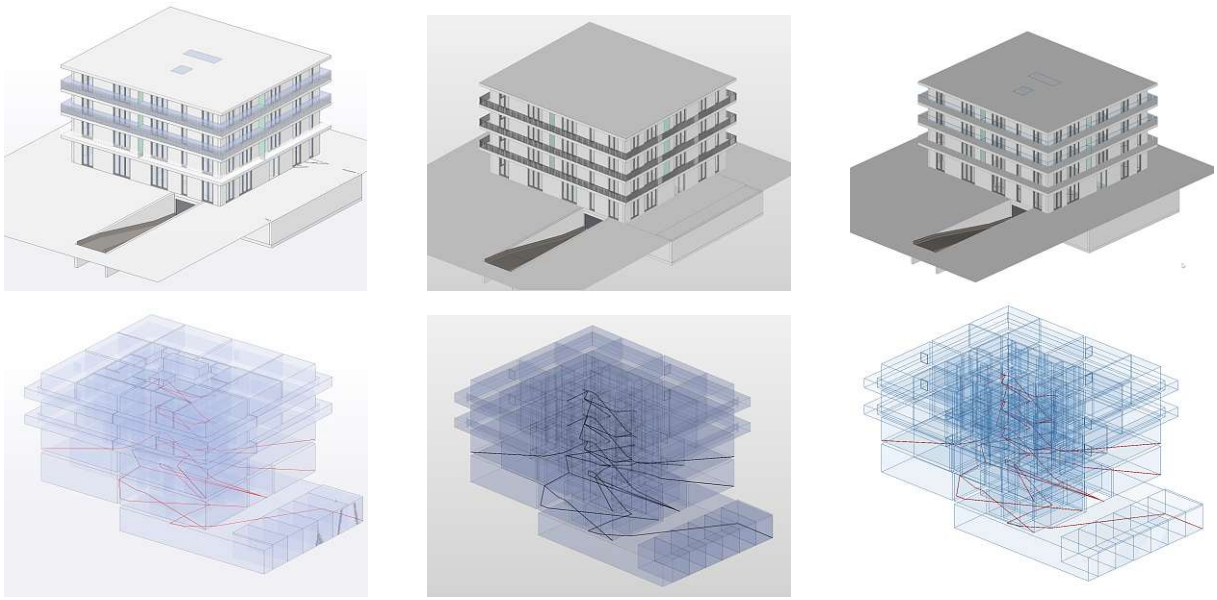


Abb. 5.1: Testmodell und generiertes Fluchtwegsmodell in den drei verwendeten Programmen. **v.l.n.r.:** TrimbleConnect, Solibri, usBIM.viewer+

Variante	Fluchtwege	Segmente	Merkmale
Annotation	18 IfcAnnotation	—	54
Proxy	—	152 IfcBuildingElementProxy	152
Property	—	152 IfcBuildingElementProxy	304
Group	18 IfcGroup	152 IfcBuildingElementProxy	188
Part	18 IfcBuildingElementProxy	152 IfcBuildingElementPart	188

Tab. 5.1: Anzahl der Objekte je Modellierungsvariante

Abbildung 5.2 zeigt das Ergebnismodell der Variante „Annotation“. Dargestellt ist ein 3D-Schnitt über dem Erdgeschoss des Testgebäudes. Der markierte Fluchtweg ist hier als eine durchgehende *IfcAnnotation* realisiert. Diese besitzt als Merkmale den Fluchtwegsnamen, die Gesamtlänge und die Personenanzahl des Fluchtweges.

Die Variante „Property“ ist in Abbildung 5.3 dargestellt. Das Modell ist in seiner Ausführung ident mit der Variante „Proxy“ und steht daher beispielhaft auch für dieses. Der Unterschied liegt nur in den Merkmalen der *IfcBuildingElementProxy*-Objekte. In dem Modell ist ersichtlich, dass einzelne Segmente selektierbar sind. Somit kann die kumulative Personenanzahl an jedem Abschnitt ausgelesen werden. In der Variante „Property“ besitzt jedes Segment eine Zugehörigkeitsliste, um Fluchtwege wieder rekonstruieren zu können. Damit sind gesamte Fluchtwegsverläufe jedoch nicht direkt auswählbar. Als Workaround können beispielhaft über das Merkmal alle Segmente eines Fluchtweges rot eingefärbt werden (siehe Abbildung 5.3).

Die Fluchtwegsmodelle der Varianten „Group“ und „Part“ sind ebenfalls in ihrer grundlegenden Darstellung und den zugeordneten Merkmalen ident. Die Segmente und gesamten Fluchtwege werden jedoch mit anderen Entities gebildet. Abbilldung 5.4 zeigt die Variante „Group“. Die Aufteilung der Merkmale auf die einzelnen Segmente und den gesamten Gruppen ermöglicht ein direktes Auslesen aller Informationen. Dargestellt sind mehrere einzelne Segmente entlang eines Fluchtweges. Diese gehören gleichzeitig mehreren Gruppen an. Die einzelnen *IfcGroup*-Elemente

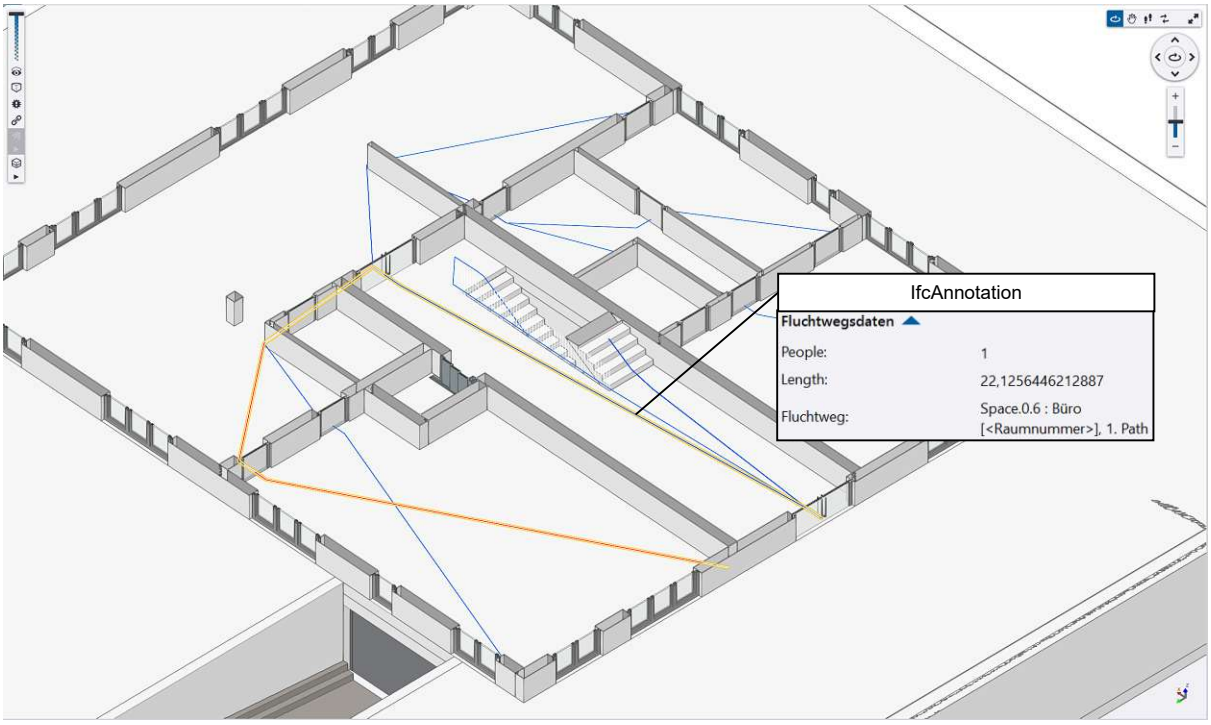


Abb. 5.2: Ergebnissmodell der Variante „Annotation“. Dargestellt ist ein Ausschnitt aus den Merkmalen der selektierten *IfcAnnotation* (in TrimbleConnect).

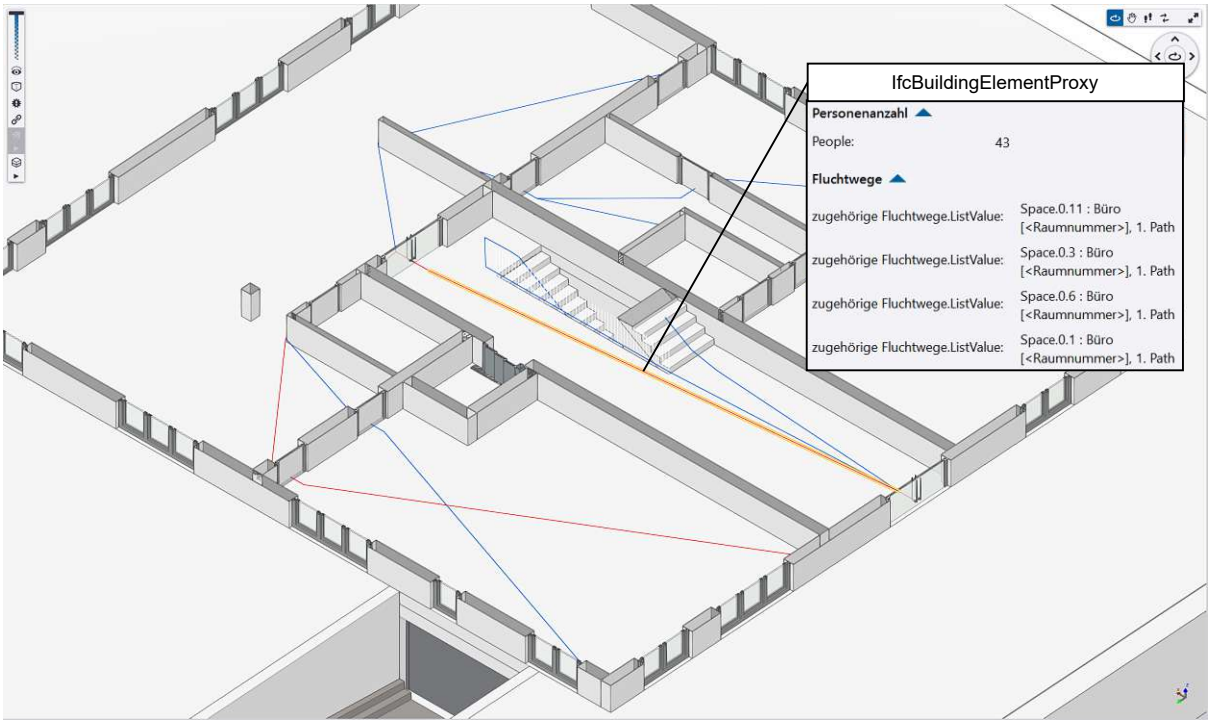


Abb. 5.3: Ergebnissmodell der Variante „Property“. Zu sehen ist ein Teil der Merkmale des selektierten *IfcBuildingElementProxy* (in TrimbleConnect).

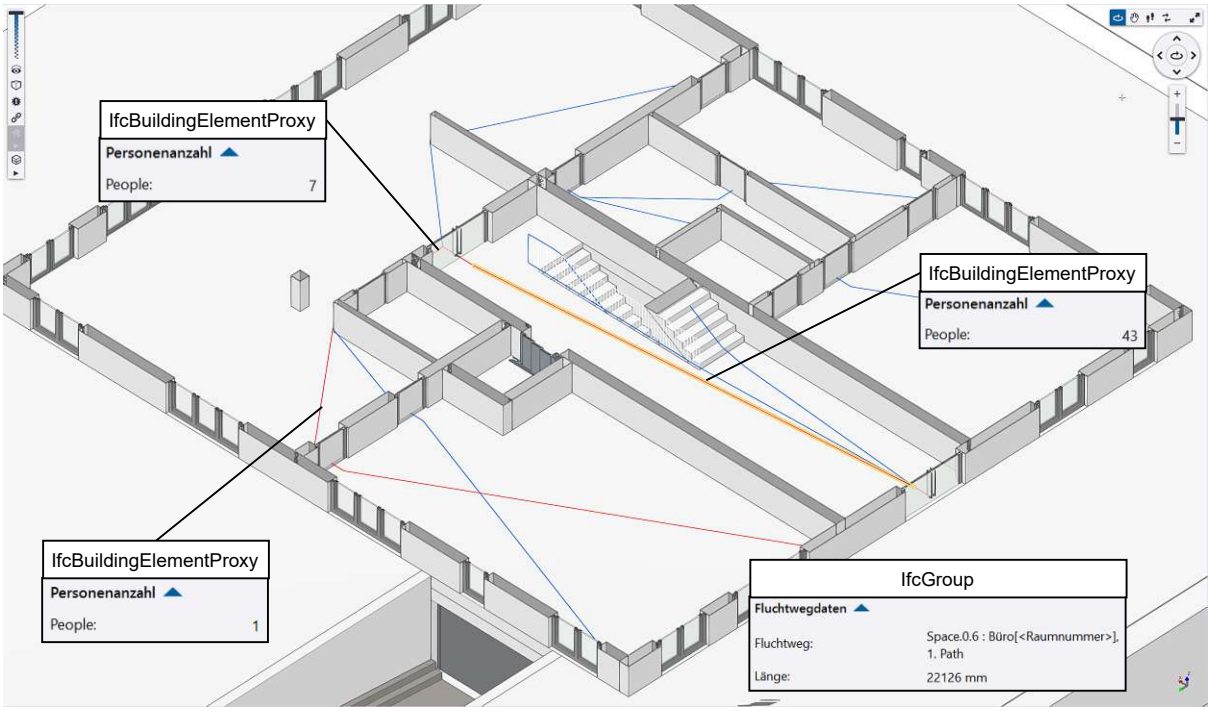


Abb. 5.4: Ergebnissmodell der Variante „Group“. Selektiert ist ein einzelnes Segment, die gesamte *IfcGroup* ist rot eingefärbt (in TrimbleConnect).

können ohne zusätzliches Filtern selektiert werden. Auch hier sind die zugewiesenen Merkmale abgebildet.

Die grundlegenden Möglichkeiten eines 3D-Fluchtwegsmodells und einer damit verbundenen 3D-Planung werden in Abbildung 5.5 ersichtlich. Mithilfe von Ansichten und Schnitten durch die Modelle ist es möglich, selbst komplizierte Fluchtwegsrouten auf einen Blick zu erfassen. Dadurch können Probleme in den Fluchtwegsrouten aufgezeigt und klar dargestellt werden. Allein das kann die Planung und Prüfung von Gebäuden maßgebend unterstützen. Die gezeigte Explosionsdarstellung ist eine Möglichkeit, die Aufteilung der Fluchtwegssegmente in dem Gebäudemodell zu zeigen. Durch die Verortung der einzelnen Segmente in den jeweiligen Stockwerken kann das Modell in getrennten Ebenen betrachtet werden. Hier zeigt sich die tatsächliche Umsetzung des Verortungsschemas aus dem Abschnitt 4.2.4.

Abseits dieser Hilfestellungen und Verbesserungen im Planungs- und Prüfungsprozess von Gebäuden können die Modelle auf verschiedenste Weise weiterverwendet werden. Abbildung 5.6 zeigt als Möglichkeit ein Fluchtwegsmodell mit adaptiven Segmentgrößen. Damit lassen sich die Größen der Personenströme visuell in einem Gebäudemodell darstellen, was die Verständlichkeit eines Fluchtwegskonzeptes verbessern kann. Auch mögliche Schlüsselstellen in den Verläufen der Fluchtwege können auf diese Weise festgestellt werden. Die gesetzlich geforderten Breiten für Flucht- und Rettungswege sind in Abhängigkeit der Personenanzahl definiert [24]. Damit könnte ein solches Modell auch bei der frühzeitigen Abklärung dieser Anforderungen eine Anwendung finden.

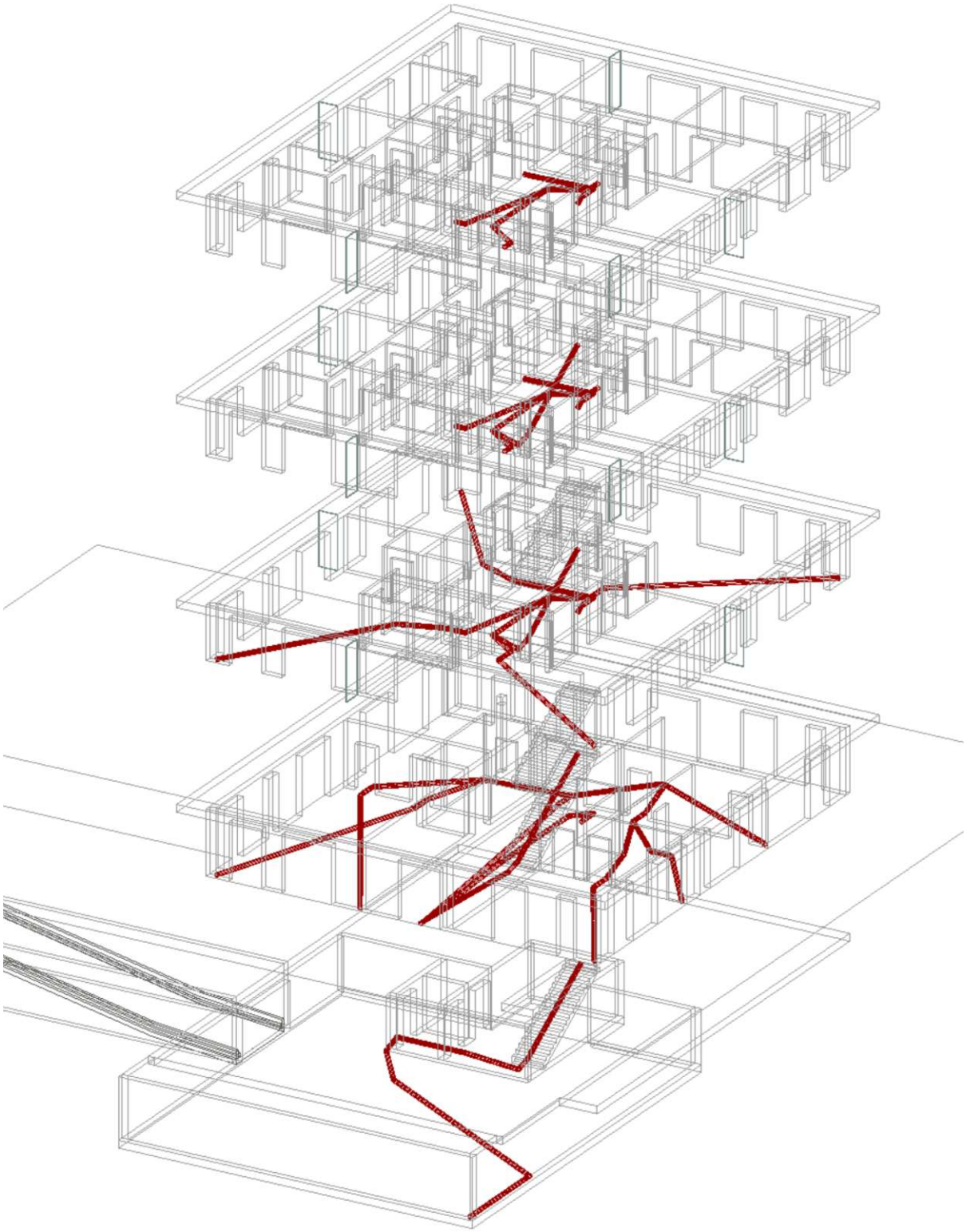
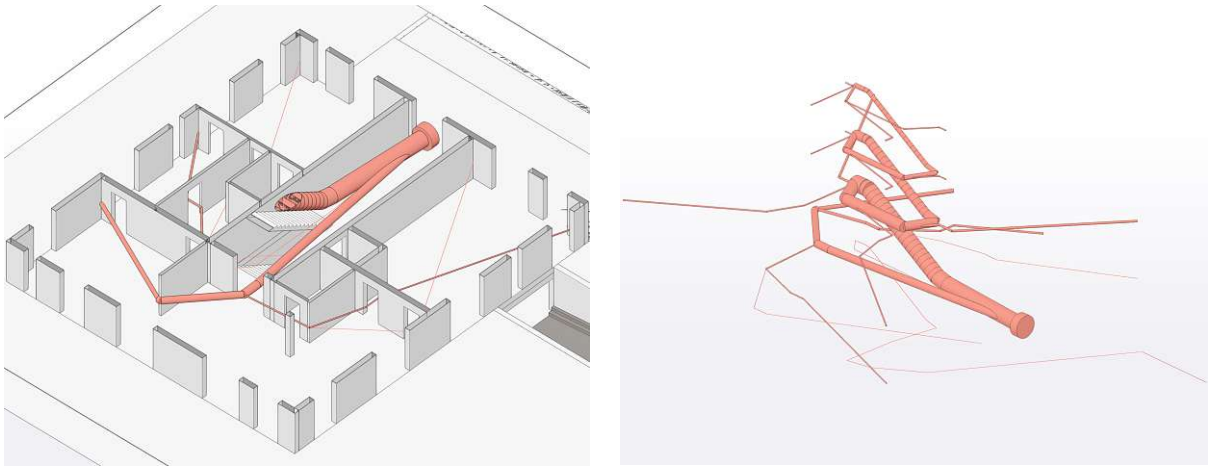


Abb. 5.5: Aufteilung der einzelnen Stockwerke eines Gebäudemodells in einer Explosionsgrafik (in usBIM.viewer+).



(a) Fluchtwegssegemente mit adaptiver Größe **(b)** Gesamtes Fluchtwegenetz des Gebäudes

Abb. 5.6: Modelle von Fluchtwegen mit größenveränderlichen Segmenten. Die Durchmesser der Segmente spiegeln dabei die kumulative Personenanzahl wieder (in TrimbleConnect).

5.2 Qualität der IFC-Dateien

Dieser Abschnitt betrachtet die Qualität der Syntax sowie des IFC-Schemas der einzelnen Modelle. Die Varianten „Annotation“, „Proxy“ und „Property“ erzeugen jeweils eine technisch einwandfreie Datei. Alle Regeln des IFC-Schemas werden eingehalten. Die komplexeren Varianten „Group“ und „Part“ weisen hier unterschiedliche Probleme auf. Abbildung 5.7 zeigt eine Übersicht der Ergebnisse des Validation Service. Diese ergeben, dass im IFC-Schema und Syntax der komplexeren Varianten Fehler vorliegen. Die Überprüfungen mit dem IfcCheckingTool liefern dieselben Erkenntnisse, wie in Abbildung 5.8 dargestellt. Nachfolgend erläutere ich die Probleme in den beiden Varianten „Group“ und „Part“.

File Name	IFC Syntax and Schema	Rules	bSDD
20230831_1516_EscapeRouteAnalysis_V5_Part.ifc	!	✓	✓
20230831_1515_EscapeRouteAnalysis_V4_Group.ifc	!	✓	✓
20230831_1514_EscapeRouteAnalysis_V3_Property.ifc	✓	✓	✓
20230831_1510_EscapeRouteAnalysis_V2_Proxy.ifc	✓	✓	✓
20230831_1508_EscapeRouteAnalysis_V1_Annotation.ifc	✓	✓	✓

Abb. 5.7: Ergebnisse der Überprüfung mit dem Validation Service von buildingSMART

Variante „Group“ verwendet die Entity *IfcGroup* zur Darstellung eines gesamten Fluchtwegs. Jede Entity, welche ein physisches Objekt beschreibt, kann in der Bauwerkshierarchie verortet werden – so zum Beispiel die einzelnen Fluchtwegssegmente selbst. Für die verwendete *IfcGroup* benutzt das Modell wie in Abbildung 3.11 die Entity *IfcRelReferencedInSpatialStructure* für die Verortung. Da die Gruppe aber kein physisches Element darstellt, kann diese auch nicht verortet werden. Wie in Abbildung 5.9 gezeigt, entsteht eine Fehlermeldung aufgrund der Definition der Entity *IfcRelReferencedInSpatialStructure*, welche ein nicht zulässiges Objekt verorten soll. Der Validation Service signalisiert, dass die Entity #3698 *IfcRelReferencedInSpatialStructure* ein

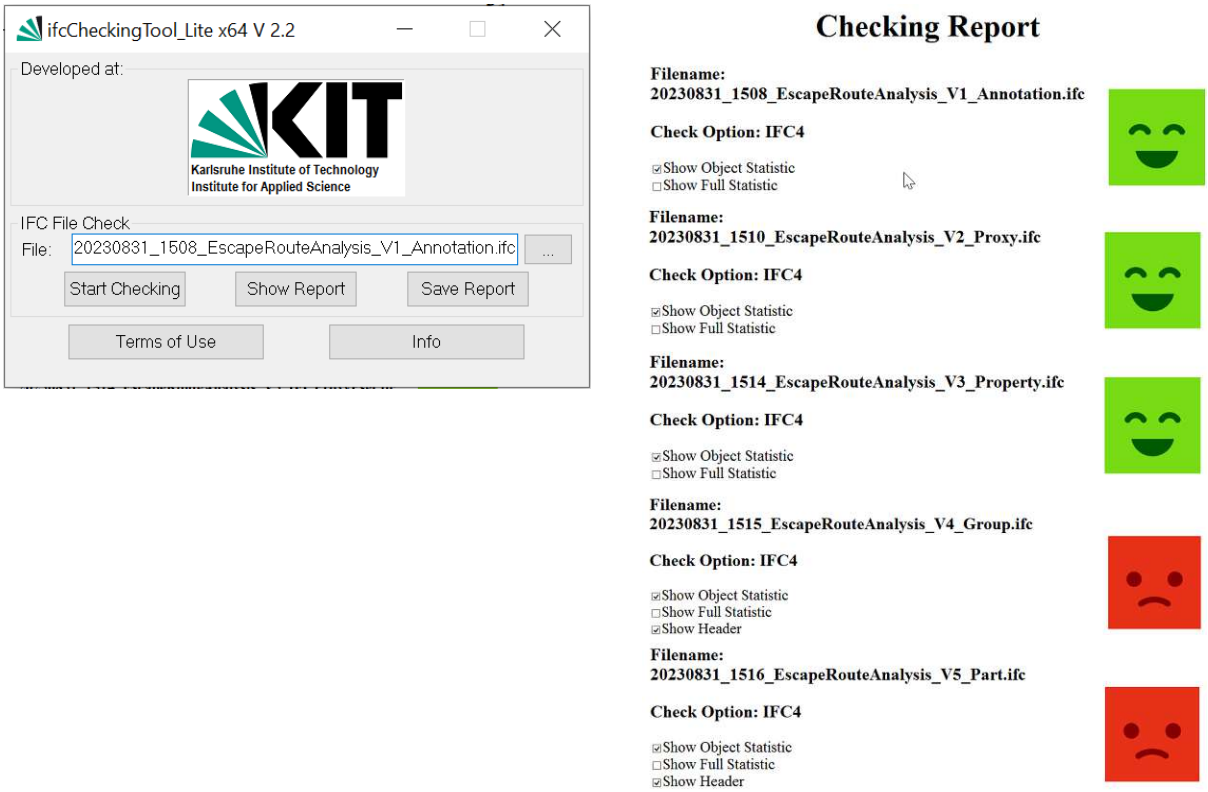


Abb. 5.8: Ergebnisse der Überprüfung mit dem KIT IfcCheckingTool

Attribut vom Typ *IfcProduct* erwartet. Die verwendete Entity #3692 *IfcGroup* ist nicht von diesem Typ und wird daher nicht akzeptiert. Dieser Umstand ist dem IFC4-Schema geschuldet, welches die Verwendung der *IfcRelReferencedInSpatialStructure* auf physische Elemente beschränkt. Das noch nicht ISO-zertifizierte IFC4.3-Schema korrigiert diese Problematik und ermöglicht die Verortung der Entity *IfcGroup*. Die aktuelle Version von IfcOpenShell unterstützt zwar das Bearbeiten von Dateien in diesem neuen Schema, kann aber selbst noch keine Modelle in diesem

Abb. 5.9: Schemafehler der Variante „Group“ im IFC4-Format

Abb. 5.10: Überprüfung der Variante „Group“ mit einer manuellen Umstellung auf das IFC4.3-Format

Dateiformat generieren. Es besteht die Möglichkeit, die Schemadefinition, welche im Header einer IFC-Datei definiert ist, mit einem simplen Textverarbeitungsprogramm zu ändern. Damit kann die Variante „Group“ als IFC4.3-Datei mit dem Validation Service überprüft werden. Hier gibt es keinen Schemafehler, wie Abbildung 5.10 zeigt.

Die Variante „Part“ verwendet keine Gruppe, um den gesamten Fluchtweg darzustellen, sondern stattdessen die Entity *IfcBuildingElementProxy*. Dieses physische Element lässt sich ohne Probleme verorten. Die Entity *IfcBuildingElementPart* stellt die einzelnen Segmente der Fluchtwege dar. Auch hier ist die Verortung möglich. Jedoch liegt die implizierte Verwendung dieser Part-Elemente darin, gemeinsam eine Baugruppe und damit genau ein Bauteil zu generieren. Die Verwendung eines einzelnen Parts in mehreren Baugruppen ist hierbei konzeptionell nicht angedacht. Wenn daher ein Segment in mehreren Fluchtwegen vorkommt, tritt bei der Zuordnung ein Fehler auf. Dieser entsteht bei der Entity *IfcRelAggregates*, welche die Zugehörigkeit der Part-Elemente steuert. Abbildung 5.11 zeigt die Fehlermeldung, welche auf die unzulässige Zuordnung eines Parts auf mehr als ein Element hinweist. Die Entity #2717 *IfcBuildingElementPart* wird über *IfcRelAggregates* mit einem Objekt verknüpft. Dadurch entsteht ein inverses Attribut. Da dieser Part von zwei Objekten verwendet wird, enthält das Attribut zwei Verweise, auf #3672 und #3679. Das Attribut darf aber nur ein Set aus maximal einem Element sein. Auch das neuere IFC4.3-Schema ändert diesen Umstand nicht.

Abb. 5.11: Schemafehler der Variante „Part“

5.3 Verwendbarkeit der IFC-Modelle

Das IFC-Format ist ein universelles Datenformat, welches jede:r Entwickler:in in einer Software integrieren kann. Daher gibt es eine Fülle an unterschiedlichen Programmen zur Verwendung dieser Dateien. IFC-Viewer besitzen einen besonders wichtigen Stellenwert, da sie meist als Freeware erhältlich sind. Mit diesen können auch ungeübte Nutzer:innen IFC-Modelle öffnen und ansehen. Dateien mit Informationen über Fluchtwege können in unterschiedlichsten Situationen

Anwendung finden. Daher sollte abseits der Fluchtwegsinformationen, welche in der Datei für eine langfristige Archivierung gespeichert sind, auch die Möglichkeit einer Verwendung der tatsächlichen Modelle mit einem Viewer existieren. Die drei Programme Solibri, TrimbleConnect und usBIM.viewer+ kommen wieder für die Beurteilung zum Einsatz.

Von einem visuellen Standpunkt unterscheidet sich nur Variante „Annotation“ von den restlichen. Die Fluchtwege sind hier als Linien ohne Volumen dargestellt. Dabei ergeben sich schnell Kollisionen mit anderen Elementen, wodurch die Fluchtwegsobjekte visuell verdeckt werden. Durch die Darstellung der gesamten Fluchtwege als ein durchgehendes Objekt ist die Verständlichkeit eines einzelnen Fluchtwegverlaufes der Variante „Annotation“ sehr gut. Gleichzeitig sind einzlene Fluchtwege nicht mehr voneinander unterscheidbar, wenn sie sich zum Beispiel bei Treppenhäusern überlagern. Abbildung 5.12 zeigt, wie sich bei einem Treppenhaus im Erdgeschoss des Gebäudes alle Fluchtwege aus einem höher Geschoss überlagern. An genau diesen Überlappungen ist das Auslesen der tatsächlichen Personenanzahl nicht möglich. Da jeder Fluchtweg nur seine eigene Personenanzahl ausgibt, müssen die einzelnen Werte mehrerer Wege manuell addiert werden. Alle Programme können das Modell verwenden und zumindest die einzelnen Informationen aller Annotationen auslesen. Hier gibt es keine technischen Einschränkungen.

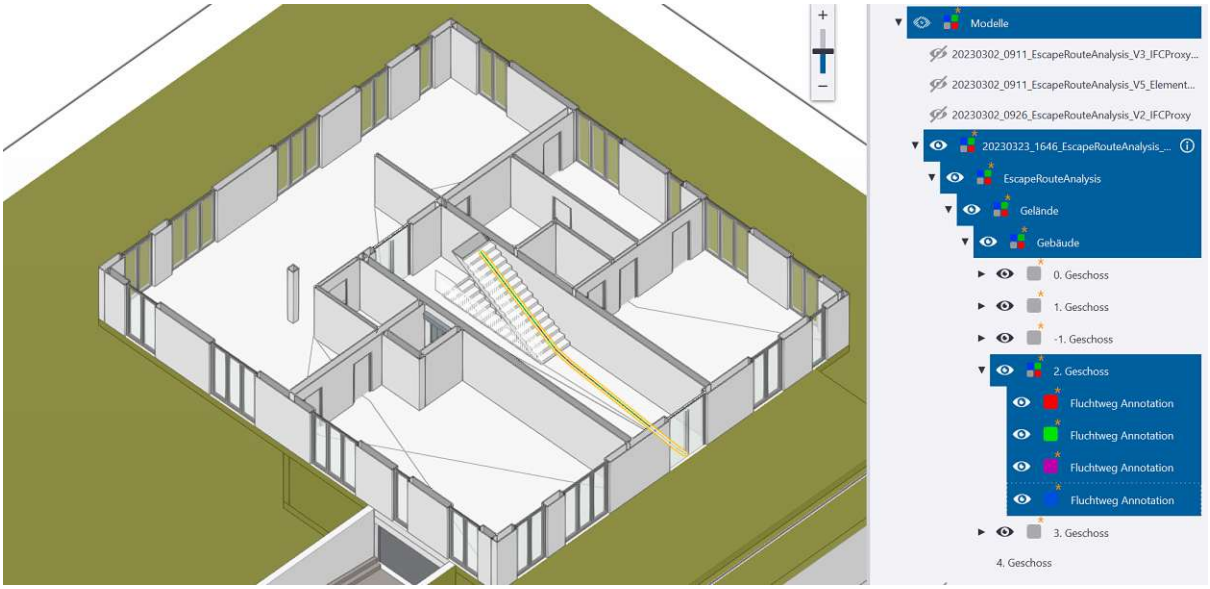


Abb. 5.12: Darstellung der Überlappung mehrere Annotationen im Bereich einer Treppe (in TrimbleConnect).

Variante „Proxy“ löst das Problem der Überlappungen mehrerer Fluchtwege durch eine Aufteilung in Segmente. Das Nachvollziehen der Personenströme ist gut möglich, da jeder Abschnitt seine eigene Personenanzahl besitzt. Durch die Verwendung von 3D-Objekten als Segmente entstehen unausweichlich Kollisionen mit Fußböden. In den Viewern hat sich daraus jedoch kein Nachteil ergeben. Die gesamten Fluchtwege sind in der Variante „Proxy“ nicht mehr rekonstruierbar. Dieser Umstand wird in der Variante „Property“ gelöst. Der Verlauf einzelner Fluchtwege kann hier über ein Merkmal rekonstruiert werden. Um einen gesamten Fluchtweg individuell gut zu erkennen, müssen alle Segmente mit einem Merkmal (z.B. Fluchtweg 1) angezeigt oder hervorgehoben werden. Diese Möglichkeit nach Merkmalen zu filtern, ist je nach Viewer unterschiedlich umgesetzt oder möglicherweise gar nicht vorhanden. Der usBIM.viewer+ bietet sehr umfangreiche Filtermöglichkeiten. Diese müssen jedoch sehr genau definiert werden, da hier jedes einzelne Merkmal aller Entities abgeglichen wird. Abbildung 5.13 zeigt das Modell

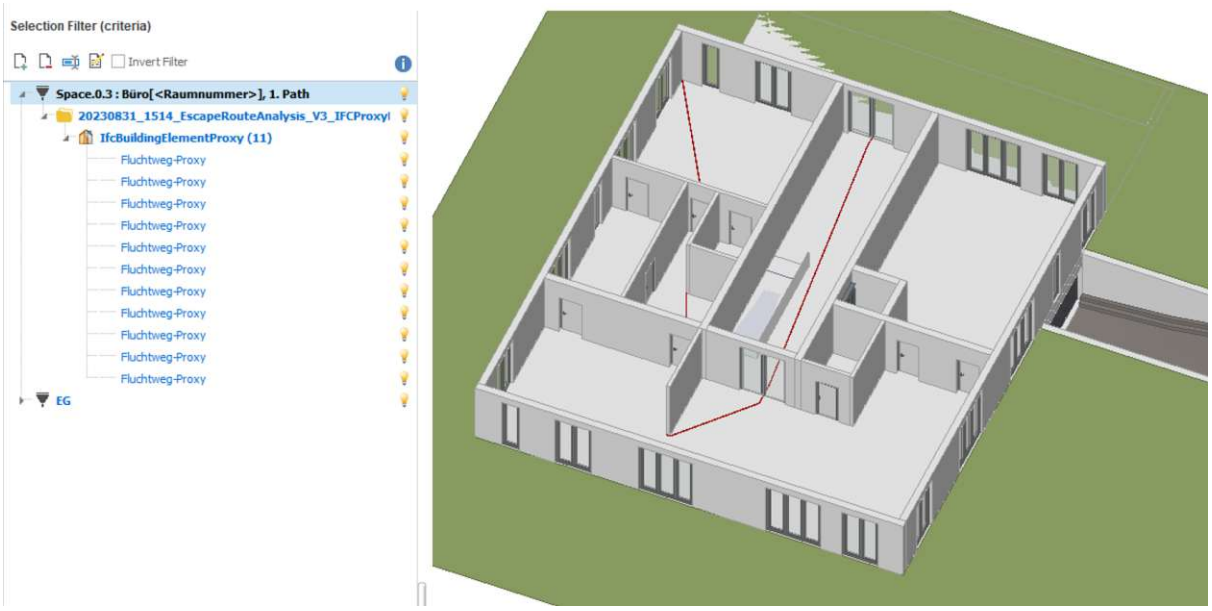


Abb. 5.13: Darstellung eines einzelnen Fluchtweges in der Variante „Property“ über das Filtern der Merkmale (in usBIM.viewer+).

mit der Filterung eines Fluchtweges im Erdgeschoss des Testgebäudes. Solibri bietet gleichwertig gute Filtermöglichkeiten. TrimbleConnect kann alle Objekte nach Begriffen durchsuchen, bietet hier aber keine Einschränkungsmöglichkeit, um diesen Filter nur auf das Merkmal für die Fluchtwegszugehörigkeiten anzuwenden. Die Anwenderfreundlichkeit ist daher nicht in allen Programmen optimal.

Variante „Group“ und „Part“ erweitern die Segmente um ein eigenes Objekt für die gesamten Fluchtwege. Die beiden Varianten ermöglichen eine sehr einfache Verwendung des Modells. Personenanzahlen können direkt am Segment ausgelesen werden. Die Informationen der gesamten Fluchtwege sind getrennt davon ebenfalls auswertbar. Bei Variante „Group“ funktioniert jedoch die Verortung der einzelnen Gruppen in der Bauwerkshierarchie mit keinem der getesteten Viewer. Auch das Modell im IFC4.3-Format wird von den Programmen noch nicht richtig verortet. Eine Ausgabe aller Fluchtwege aus einem bestimmten Stockwerk ist daher zurzeit nicht möglich. Abbildung 5.14 zeigt das reine Fluchtwegsmodell der Variante „Group“ im Programm Solibri. Im Strukturbaum ist klar ersichtlich, dass die Gruppen separat von der Bauwerksstruktur existieren. Dieser Schemafehler der Variante „Group“ wird jedoch von keinem der Viewer direkt als Fehler angezeigt, sondern ignoriert. Gleiches trifft auf den in Abschnitt 5.2 beschriebenen Schemafehler der Variante „Part“ zu. Es besteht die Möglichkeit, dass andere Viewer diese Fehler erkennen bzw. die Datei daher von diesen nicht einwandfrei verwendet werden kann.

5.4 Bewertung der Fluchtwegsmodelle

Das IFC-Format bietet bereits einen großen Umfang an Objekten und Möglichkeiten. Gleichzeitig ist der Facettenreichtum von Gebäuden sehr groß. Es ist daher unmöglich, im Rahmen eines Dateiformates für alle gewünschten Objekte vorgefertigte Lösungen anzubieten. Einer dieser Fälle sind Fluchtwege, welche auf die unterschiedlichsten Weisen realisiert werden können. Die entwickelte Modellierung dieser nicht vordefinierten Elemente soll einerseits die Informationen richtig abspeichern und gleichzeitig im IFC-Schema einwandfrei funktionieren. Die fünf unter-

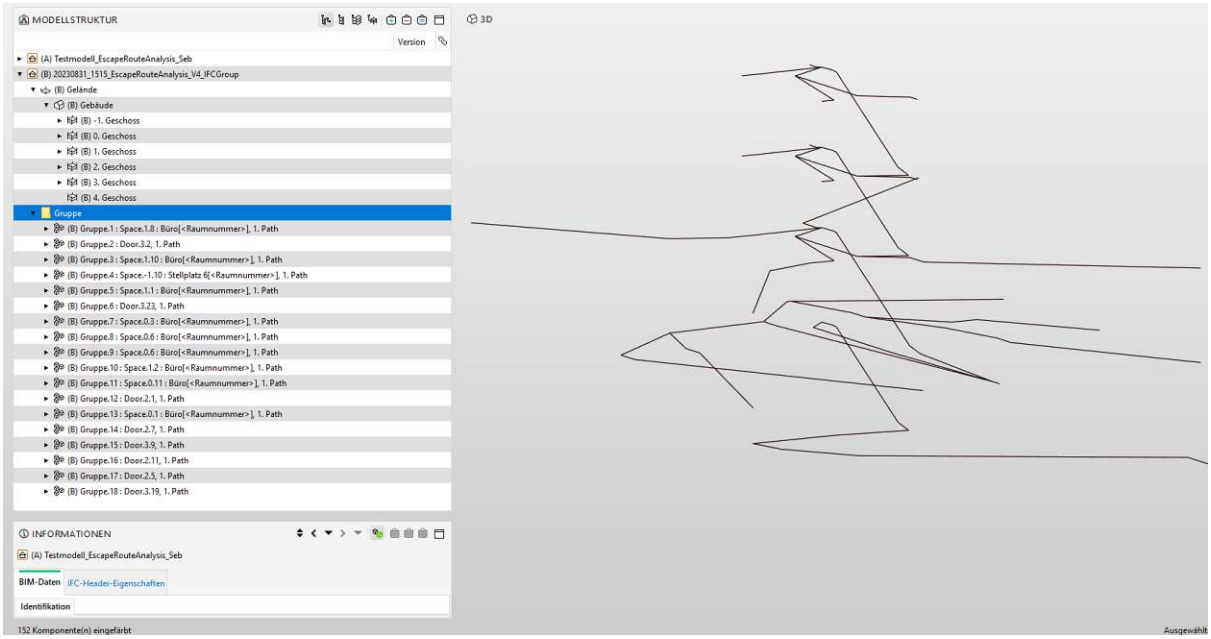


Abb. 5.14: Dateistruktur der Variante „Group“. Die Gruppen sind nicht in der Bauwerksstruktur verortet (in Solibri).

suchten Varianten besitzen alle unterschiedliche Vor- und Nachteile. Es kann pauschal keine optimale Modellierung identifiziert werden. Die Varianten zeigen jedoch, dass für unterschiedliche Anwendungsfälle verschiedene Modelle die jeweils beste Lösung darstellen können.

Variante „Annotation“ stellt die einfachst-mögliche Modellierung eines Fluchtweges dar. Sie gibt den Pfad durch das Gebäude als eine reine Linie wieder, bietet aber keine zusätzlichen Funktionen. Für die Darstellung eines einzelnen Fluchtweges in einem Gebäude ist diese Variante aufgrund ihrer simplen Ausführung ideal geeignet. Bei mehreren Fluchtwegen ist diese Modellierung jedoch nicht mehr sinnvoll. Die Variante „Proxy“ kann mehrere Fluchtwege ohne Überlagerungen darstellen. Das Auslesen der Personenströme ist sehr einfach möglich, da jedes Segment diese Information direkt besitzt. Da jedoch die einzelnen Fluchtwegsverläufe nicht mehr rekonstruierbar sind, erscheint diese Methode als nicht sinnvoll. Einzig für den sehr speziellen Anwendungsfall, dass die reinen Größen der Personenströme durch ein Gebäude angezeigt werden sollen, könnte sie geeignet sein. Um die Fluchtwegsverläufe und damit alle Informationen zu erhalten, ist die Variante „Property“ denkbar. Diese Methode ermöglicht es rein theoretisch, alle Fluchtwegsinformationen wieder aus der erstellten IFC-Datei auszulesen. Es entstehen jedoch zwei markante Nachteile bei dieser Methode. Einerseits sind die gesamten Fluchtwegslängen nicht direkt abgespeichert, sondern nur aus den Längen der zusammengehörigen Segmente errechenbar. Andererseits kann die schlechte Anwenderfreundlichkeit als negativer Aspekt gesehen werden. Da die Verläufe der einzelnen Fluchtwege nur über ein Merkmal definiert sind, ist ein schnelles und einfaches Auslesen der gesamten Fluchtwege in einem Viewer nicht gegeben. Variante „Group“ und „Part“ stellen eine konzeptionell idente Lösung dar. Beide Möglichkeiten enthalten alle Informationen über die Fluchtwege. Durch die Verwendung getrennter Entities für die einzelnen Segmente und die gesamten Fluchtwege sind die Modelle sehr gut verständlich. Die Syntaxfehler in den Dateien stellen jedoch eine unsaubere Modellierung dar. Dieser Umstand kann zu unvorhergesehenen Problemen führen. Mit der fortschreitenden Implementierung und Anwendung des IFC4.3-Schemas ergibt sich jedoch bei der Variante „Group“ ein fehlerfreies Modell.

Tabelle 5.2 zeigt die zusammengefasste Bewertung der unterschiedlichen Lösungen. Die Variante „Property“ bietet zum Zeitpunkt dieser Arbeit eine technisch mögliche, jedoch aufgrund der nicht vollständigen Daten und der umständlichen Darstellung der gesamten Fluchtwegsverläufe keine zufriedenstellende Lösung. Im Datenumfang und der Handhabung des Modells stellt „Part“ die bevorzugte Fluchtwegsdatei dar. Jedoch ist der Schemafehler in den verwendeten Entities aus dem Gesichtspunkt der Datenarchivierung nicht akzeptabel. Mit dem sich immer stärker etablierenden IFC4.3-Standard bietet die Variante „Group“ daher rundum die beste Lösung. In dieser werden alle Informationen der Fluchtwegsanalyse sauber abgespeichert und sind auch für eine direkte Verwendung gut einsetzbar. Das daraus entstehende IFC-Modell erfüllt die Anforderungen eines Fluchtwegsmodells am besten, und kann damit als sinnvollste Modellierungsvariante betrachtet werden. Die Verwendung der Entity *IfcAnnotation* anstelle von *IfcBuildingElementProxy* ist bei allen Varianten denkbar. Diese Entscheidung ist eher kosmetischer Natur. Es ist jedoch zu erwarten, dass ein Volumenkörper, und damit das Proxy-Element, als Platzhalter verständlicher ist.

Variante	Semantik und Schema	Vollständigkeit der Daten	Verwendbarkeit des IFC-Modells
Annotation	+	○	○
Proxy	+	–	–
Property	+	○	○
Group	–	+	○
Group im IFC4.3-Format	+	+	+
Part	–	+	+

Tab. 5.2: Vergleich der getesteten Modellierungsvarianten

5.5 Fazit zur IfcOpenShell-Bibliothek

Die IfcOpenShell-Bibliothek ermöglicht eine einfache und direkte Automatisierung der Bearbeitung von IFC-Dateien. Sie wurde mit diesem Grundgedanken aufgebaut und kann daher die vollen Möglichkeiten des IFC-Schemas verwenden. Die angewendeten Befehle entsprechen in Schreibweise und Verwendung praktisch direkt dem IFC-Datenschema. Ein Skript ist daher mit ausreichenden Kenntnissen der IFC-Struktur verständlich. Hier jedoch liegt eine der größten Limitationen dieser Bibliothek.

Das IFC-Datenschema bietet eine Vielzahl an Entitäten, welche sehr viele Aufgabengebiete abdecken können. Damit steigt jedoch auch die Komplexität der Funktionen enorm. Der Umgang erfordert ein hohes Verständnis der zugrundeliegenden IFC-Datenstruktur. Dadurch ist auch die IfcOpenShell-Bibliothek mit einer steilen Lernkurve verbunden. IfcOpenShell kann nicht erkennen, welche IFC-Befehle für die Erreichung eines Zieles notwendig sind. Es sind keine ausreichenden Funktionen umgesetzt, welche verhindern, dass eine IFC-Datei generiert wird, welche semantisch falsch und somit unbrauchbar ist. Die Richtigkeit der Verknüpfungen einzelner Entitäten zueinander kann die Bibliothek nicht prüfen. Es werden jedoch die Entitäten selbst auf ihre Vollständigkeit untersucht. IfcOpenShell verhindert, dass Objekte mit fehlerhaften oder fehlenden Attributen in die IFC-Datei geschrieben werden. Damit übernimmt es diese Schwierigkeit beim Umgang mit IFC-Dateien.

Python ist als Grundlage für IfcOpenShell eine sehr anwenderfreundliche und weit verbreitete Programmiersprache. Das ermöglicht die einfache Integration der Bibliothek in andere automatisierte Prozesse. Zusätzlich öffnet es den Zugang zum IFC-Format für viele Entwickler.

IfcOpenShell besitzt außerdem eine aktive Community der Entwickler und Anwender. Diese kann bei Projekten wertvollen Input und Unterstützung geben. Gleichzeitig ist jedoch die offizielle Dokumentation der Bibliothek begrenzt und deckt nicht jeden Anwendungsfall und jede technische Möglichkeit ab, die IfcOpenShell bietet.

Zusammenfassend lässt sich sagen, dass die IfcOpenShell-Bibliothek ein umfangreiches und leistungsfähiges Werkzeug im Umgang mit IFC-Dateien darstellt. Das Programm kann durch seine nahtlose Pythonintegration sehr gut verwendet und in andere Prozesse eingebaut werden. Ein fundiertes Wissen über das IFC-Schema sowie eine gewisse Einarbeitungszeit sind jedoch unumgänglich, um die Möglichkeiten der Bibliothek voll auszuschöpfen.

Kapitel 6

Zusammenfassung und Ausblick

Im Zentrum dieser Arbeit steht die grundlegende Suche nach einer sinnhaften Implementierung von Fluchtwegsinformationen in einer IFC-Datei. Zu Beginn wird diese Anforderung durch zwei Forschungsfragen näher konkretisiert: Wie kann ein Fluchtweg im Rahmen des IFC-Schemas dargestellt werden und wie kann dieser Prozess automatisiert werden? Im Zuge der Beantwortung habe ich die Möglichkeiten der verschiedenen IFC-Entities betrachtet und damit fünf unterschiedliche Modellierungsvarianten erarbeitet. Aus diesen hat sich, mit Blick auf die weitere Entwicklung des IFC-Formates, Variante „Group“ als beste Lösung etabliert. Variante „Annotation“ kann mehrfach verwendete Fluchtwegsabschnitte nicht richtig kombinieren. „Proxy“ verliert die gesamten Fluchtwegsverläufe. „Property“ behält diese, speichert jedoch die Fluchtwegslängen nicht ab. „Part“ besitzt einen nicht lösbaren Schemafehler.

Bei Variante „Group“ werden gesamte Fluchtwege durch ein Gebäude in einzelne Segmente unterteilt. Diese erhalten als Merkmale die kumulative Personenanzahl aller Fluchtwege, die diesen Abschnitt verwenden. In der IFC-Datei stellen Objekte der Entity *IfcBuildingElementProxy* diese Segmente dar. Gruppen fassen diese wieder zu den gesamten Fluchtwegen zusammen. Hierfür wird die Entity *IfcGroup* verwendet. Diese Objekte besitzen als Merkmale den Namen, beziehungsweise die Kennung, sowie die gesamte Länge des jeweiligen Fluchtweges. Diese Modellierung speichert alle geforderten Informationen in einer IFC-Datei ab. Zusätzlich können Benutzer:innen das Modell in einem IFC-Viewer direkt öffnen und die relevanten Daten auf einfache Weise auslesen. Variante „Group“ vereint alle diese Vorteile miteinander. Einzig der Schemafehler im IFC4-Format kann als Limitation gesehen werden. Mit der zunehmenden Adaptierung des IFC4.3-Schemas wird dieser Umstand jedoch gelöst.

Die automatisierte Generierung der einzelnen Modelle erfolgt mithilfe der Programmiersprache Python und der frei verfügbaren IfcOpenShell-Bibliothek. Diese ermöglicht es, IFC-Dateien in einer Python Entwicklungsoberfläche zu bearbeiten. Eine Eingabedatei in Form einer JSON-Datei liefert die Ergebnisse einer vorangestellten Fluchtwegsberechnung. Aus diesen Informationen wird in einem neuen IFC-Modell eine Bauwerkshierarchie generiert, in der die einzelnen Fluchtwege lagerichtig verortet werden. Für jedes Fluchtwegssegment erstellt das Programm neue geometrische Objekte. Die Fluchtwegsdaten werden über Merkmale mit den Objekten verknüpft. Schlussendlich werden die einzelnen Segmente je nach Modellierungsvariante gruppiert. Das Programm speichert das fertige Modell als eine eigenständige IFC-Datei ab. Durch diesen Prozess kann die Modellgenerierung vollautomatisch ablaufen. Die IfcOpenShell-Bibliothek zeigt sich hier als leistungsfähiges Werkzeug, wobei jedoch ein umfangreiches Wissen der IFC-Datenstruktur vorausgesetzt wird.

Die Integration von Fluchtwegsdaten in einem BIM-Modell stellt einen weiteren Baustein der Möglichkeiten einer modellbasierten Planung dar. Mit der immer stärkeren Adaptierung der BIM-Methode in der Planung wird die Bedeutung des digitalen Gebäudezwillings über den gesamten Lebenszyklus eines Bauwerkes zunehmen. Derzeit enthält das erarbeitete Fluchtwegsmodell die exakt generierten Fluchtwegsdaten. Diese sind für eine behördliche Überprüfung und für den Nachweis der Einhaltung aller gesetzlichen Vorschriften gut geeignet. Gleichzeitig ermög-

licht diese Abspeicherung der Daten eine langfristige, sichere Archivierung der Informationen. Das ISO-zertifizierte IFC-Format stellt als frei zugängliches Datenformat hierfür die optimale Grundlage dar. Für eine weitere Anwendung des Modells aus Sicht des Gebäudebetreibers könnte eine andere Modellierung die bessere Lösung darstellen. Nachfolgende Arbeiten könnten aus den Fluchtwegsdaten visuell noch ansprechendere Darstellungen entwickeln, welche direkt für Visualisierungen im Gebäudebetrieb anwendbar wären. Ebenfalls stellt eine anwenderfreundliche Implementierung des entwickelten Programmcodes eine interessante Folgearbeit dar. Durch eine eigenständige Software oder die Integration in einzelne 3D-Modellierungsprogramme über eine API könnten Fluchtwegsmodelle im IFC-Format für viele Planungsbüros einfach realisierbar werden. Solche Möglichkeiten würden die zukünftige Planung, Überprüfung und das spätere Verständnis von Fluchtwegen in Gebäuden verbessern und damit den hohen Sicherheitsstandard im Bauwesen weiterhin garantieren.

Literatur

[1] ACCA software S.p.A. *usBIM.viewer+*. 2023. URL: https://www.accasoftware.com/de/ifc-viewer#usbim-viewer (Zugriff am 13. 09. 2023).

[2] K. Aengenvoort und M. Krämer. „BIM im Betrieb von Bauwerken“. In: *Building Information Modeling: Technologische Grundlagen und industrielle Praxis*. Hrsg. von A. Borrmann, M. König, C. Koch und J. Beetz. Wiesbaden: Springer Fachmedien Wiesbaden, 2021, S. 611–644. DOI: 10.1007/978-3-658-33361-4_32.

[3] A. Borrmann, M. König, C. Koch und J. Beetz. „Die BIM-Methode im Überblick“. In: *Building Information Modeling: Technologische Grundlagen und industrielle Praxis*. Hrsg. von A. Borrmann, M. König, C. Koch und J. Beetz. Wiesbaden: Springer Fachmedien Wiesbaden, 2021, S. 1–31. DOI: 10.1007/978-3-658-33361-4_1.

[4] buildingSMART International. *Annex B Alphabetical listings - Entities*. 2023. URL: https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/annex-b1.html (Zugriff am 04. 05. 2023).

[5] buildingSMART International. *IFC 4.3 documentation*. 2023. URL: https://ifc43-docs.standards.buildingsmart.org/ (Zugriff am 17. 09. 2023).

[6] buildingSMART International. *IfcAnnotation*. 2023. URL: https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/lexical/IfcAnnotation.htm (Zugriff am 20. 05. 2023).

[7] buildingSMART International. *IfcBuildingElementProxy*. 2023. URL: https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/lexical/IfcBuildingElementProxy.htm (Zugriff am 17. 05. 2023).

[8] buildingSMART International. *IfcPropertySingleValue*. 2023. URL: https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/lexical/IfcPropertySingleValue.htm (Zugriff am 20. 05. 2023).

[9] buildingSMART International. *Industry Foundation Classes (IFC)*. 2023. URL: https://technical.buildingsmart.org/standards/ifc/ (Zugriff am 21. 09. 2023).

[10] buildingSMART International. *terms, definitions and abbreviated terms*. 2023. URL: https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/content/terms_and_definitions.htm (Zugriff am 14. 09. 2023).

[11] buildingSMART International. *Validation Service*. 2023. URL: https://technical.buildingsmart.org/services/validation-service/ (Zugriff am 30. 08. 2023).

[12] C. Eastman, J.-m. Lee, Y.-s. Jeong und J.-k. Lee. „Automatic rule-based checking of building designs“. In: *Automation in Construction* 18(8) (2009), S. 1011–1033. DOI: 10.1016/j.autcon.2009.07.002.

[13] C. C. Eichler, C. Schranz, T. Krischmann und H. Urban. *BIMcert Handbuch, Grundlagenwissen openBIM*. Ausgabe 2023. Niederfrohna: Mironda-Verlag, 2023. ISBN: 978-3-96063-052-4.

[14] S. Fischer, C. Schranz, H. Urban und D. Pfeiffer. „Automation of escape route analysis for BIM-based building code checking“. In: *Automation in Construction* 156 (2023), S. 105092. DOI: 10.1016/j.autcon.2023.105092.

[15] S. Fischer, C. Schranz, H. Urban und D. Pfeiffer. *Custom Escape Route Test Model in IFC format*. Juli 2023. DOI: 10.48436/ra5g9-tbb65.

[16] IfcOpenShell Contributors. *IfcOpenShell 0.7.0 documentation*. 2023. URL: https://blenderbim.org/docs-python/ifcopenshell.html (Zugriff am 17. 06. 2023).

[17] *ISO 10303-11:2004 – Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual*. Englisch. International Organization for Standardization, Nov. 2004.

[18] *ISO 16739-1:2018 – Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries — Part 1: Data schema*. Englisch. International Organization for Standardization, Nov. 2018.

[19] M. Kannala. „Escape Route Analysis Based on Building Information Models: Design and Implementation“. MSc thesis. Helsinki, Finland: Department of Computer Science und Engineering, Helsinki University of Technology, 2005. URL: http://mattikannala.fi/projects/masters-thesis/Escape-Route-Analysis-Based-on-Building-Information-Models-Design-and-Implementation.pdf (Zugriff am 02. 11. 2023).

[20] Karlsruher Institut für Technologie, Institut für Automation und angewandte Informatik. *IfcCheckingTool*. 2023. URL: https://www.iai.kit.edu/1649.php (Zugriff am 30. 08. 2023).

[21] T. Krischmann, H. Urban und C. Schranz. „Entwicklung eines openBIM-Bewilligungsverfahrens“. In: *Bauingenieure* 95(9) (2020), S. 335–344. DOI: 10.37544/0005-6650.

[22] J.-K. Lee, C. M. Eastman, J. Lee, M. Kannala und Y.-S. Jeong. „Computing Walking Distances within Buildings Using the Universal Circulation Network“. In: *Environment and Planning B: Planning and Design* 37(4) (2010), S. 628–645. DOI: 10.1068/b35124.

[23] A. Merschbacher. „Bedarf an Fluchtwegen“. In: *Flucht- und Rettungswege: Anforderungen behinderter Menschen an die Bewältigung von Notfällen*. Wiesbaden: Springer Fachmedien Wiesbaden, 2021, S. 1–5. DOI: 10.1007/978-3-658-32845-0_1.

[24] *OIB-Richtlinie 2 – Brandschutz*. Deutsch. Wien: Österreichisches Institut für Bautechnik, Apr. 2019.

[25] *OIB-Richtlinie 4 – Nutzungssicherheit und Barrierefreiheit*. Deutsch. Wien: Österreichisches Institut für Bautechnik, März 2019.

[26] *OIB-Richtlinien Begriffsbestimmungen*. Deutsch. Wien: Österreichisches Institut für Bautechnik, Apr. 2019.

[27] Open Source Initiative. *GNU Lesser General Public License version 3*. 2007. URL: https://opensource.org/license/lgpl-3-0/ (Zugriff am 17. 06. 2023).

[28] Python Software Foundation. *Generate temporary files and directories*. 2023. URL: https://docs.python.org/3/library/tempfile.html (Zugriff am 04. 05. 2023).

[29] Python Software Foundation. *JSON encoder and decoder*. 2023. URL: https://docs.python.org/3/library/json.html (Zugriff am 30. 04. 2023).

[30] R. Sacks, C. Eastman, G. Lee und P. Teicholz. *BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility Managers*. Hoboken: John Wiley & Sons, Inc., Aug. 2018. ISBN: 978-1-119-28753-7.

[31] F. Schnabel. „Automated generation of escape route models using IFC.js“. Bachelorarbeit. Wien: TU Wien, 2023.

[32] Solibri INC., A Nemetschek Company. *Solibri*. 2023. URL: https://www.solibri.com/de/ (Zugriff am 13. 09. 2023).

[33] Solibri INC., A Nemetschek Company. *Solibri Office*. 2023. URL: https://www.solibri.com/de/solibri-office (Zugriff am 23. 10. 2023).

[34] Stadt Wien. *BRISE-VIENNA - Projektbeschreibung*. 2020. URL: https://digitales.wien.gv.at/projekt/brisevienna/ (Zugriff am 22. 09. 2023).

[35] Trimble Inc. *Trimble Connect*. 2023. URL: https://connect.trimble.com/ (Zugriff am 13. 09. 2023).

[36] UIA – Urban Innovative Actions. *BRISE-Vienna – Building Regulations Information for Submission Envolvement*. 2023. URL: https://www.uia-initiative.eu/en/uia-cities/vienna-call4 (Zugriff am 02. 11. 2023).